L Number	Hits	Search Text	DB	Time stamp
2	0	prox$4 near10 cluster$4 near10 rout$4 near10 request$1 near10 (document$1 page$1 resourc$3 file$1) near10 server$1 and @ad<20000803 and 709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/12/10 11:57
3	17355	prox$4 near10 cluster$4 near10 rout$4 near10 request$1 near10 (document$1 page$1 resourc$3 file$1) near10 server$1 @ad<20000803 and 709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/12/10 11:57
1	13	prox$4 near10 rout$4 near10 request$1 near10 (document$1 page$1 resourc$3 file$1) near10 server$1 and @ad<20000803 and 709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/12/10 11:57
6	110	(forward$4 redirect$4) near10 request$1 near10 (server$1 prox$4) near10 (URL$1) and @ad<20000803	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/12/10 11:57
7	174	(forward$4 redirect$4) near10 request$1 near10 (server$1 prox$4) near10 (address$3) and @ad<20000803	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/12/10 11:57
4	25	(forward$4 redirect$4) near10 request$1 near10 (prox$4) near10 (URL$1) and @ad<20000803 and 709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/12/10 11:57
5	82	(forward$4 redirect$4) near10 request$1 near10 (server$1 prox$4) near10 (URL$1) and @ad<20000803 and 709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/12/10 11:57
8	19	choos$4 near10 (prox$4 server$1) near10 address$4 near10 client$1 and @ad<20000803	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/12/10 11:57
9	15	choos$4 near10 server$1 near10 address$4 near10 client$1 and @ad<20000803	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/12/10 11:57
10	31	compar$4 near10 address$4 near10 list$3 near10 server$1 and @ad<20000803	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/12/10 11:57
-	0	select$4 near10 (proxy or proxies) near10 highest near10 speed$1	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/25 13:04
-	2	select$4 near10 (proxy or proxies) near10 speed$1	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/25 13:05
-	170	select$4 near10 (server$1) near10 speed$1	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/25 13:06

-	101	select$4 near10 (server$1) near10 speed$1 and @ad<20000803	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/25 13:06
-	38	select$4 near10 (server$1) near10 speed$1 and @ad<20000803 and 709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/25 14:06
-	0	select$4 near10 (proxy or proxies) near10 speed$1 and @ad<20000803 and 709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/25 13:09
-	1	select$4 near10 (server$1) near10 speed$1 and @ad<20000803 and 709/226.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/25 14:07
-	2	select$4 near10 (server$1) near10 speed$1 and @ad<20000803 and 709/229.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 08:45
-	7	select$4 near10 (server$1 proxy proxies) near10 destination$1 and @ad<20000803 and 709/226.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/25 14:27
-	73	forward$4 near10 request$1 near10 (resource$1 page$1 webpage$1 website$1) near10 (server$1 proxy proxies) and @ad<20000803 and 709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 08:33
-	98	select$4 near10 (server$1 proxy proxies) near10 destination$1 and @ad<20000803 and 709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/25 14:29
-	11	forward$4 near10 request$1 near10 (resource$1 page$1 webpage$1 website$1) near10 (server$1 proxy proxies) and @ad<20000803 and 709/226.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 08:14
-	0	forward$4 near10 request$1 near10 (check$4 near10 table$1) near10 (server$1 proxy proxies) and @ad<20000803 and 709/226.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 08:14
-	0	forward$4 near10 request$1 near10 (table$1) near10 (server$1 proxy proxies) and @ad<20000803 and 709/226.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 08:15
-	2	forward$4 near10 (table$1) near10 (server$1 proxy proxies) and @ad<20000803 and 709/226.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 08:18
-	39	forward$4 near10 (table$1) near10 (server$1 proxy proxies) and @ad<20000803 and 709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 08:20
-	8	forward$4 near10 (table$1) near10 (proxy proxies) and @ad<20000803 and 709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 08:22

-	29	forward$4 near10 request$1 near10 (resource$1 page$1 webpage$1 website$1) near10 (server$1 proxy proxies) and @ad<20000803 and 709/217-219.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 08:33
-	11	forward$4 near10 request$1 near10 (resource$1 page$1 webpage$1 website$1) near10 (server$1 proxy proxies) and @ad<20000803 and 709/226.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 08:36
-	11	compar$4 near10 address$3 near10 table$1 near10 (proxy proxies server$1) and @ad<20000803 and 709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 08:43
-	1	compar$4 near10 address$3 near10 table$1 near10 (proxy proxies)and @ad<20000803 and 709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 08:44
-	0	(proxy proxies) near10 table$1 near10 address$2 near10 forward$3 and @ad<20000803 and 709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 08:46
-	25	(proxy proxies) near10 table$1 near10 address$2 and @ad<20000803 and 709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 08:55
-	4	(proxy proxies) near10 table$1 same default and @ad<20000803 and 709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 08:56
-	28	compar$4 near10 address$5 near10 table$1 near10 (proxy server$1) and @ad<20000803	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 09:14
-	8	compar$4 near10 server$1 near5 IP near10 (proxy proxies) and @ad<20000803	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 09:16
-	0	compar$4 near10 resource$1 near5 IP near10 (proxy proxies) and @ad<20000803	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 09:17
-	0	compar$4 near10 destination near5 IP near10 (proxy proxies) and @ad<20000803	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 09:16
-	13	compar$4 near10 IP near10 (proxy proxies) and @ad<20000803	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 09:17
-	195	forward$4 near10 request$1 near10 (proxy proxies) near10 (address$3 URL$1 server$1 destination$1) and @ad<20000803	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 09:31
-	134	forward$4 near10 request$1 near10 (proxy proxies) near10 (address$3 URL$1 server$1 destination$1) and @ad<20000803 and 709/$.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 09:31

-	14	forward$4 near10 request$1 near10 (proxy proxies) near10 (address$3 URL$1 server$1 destination$1) and @ad<20000803 and 709/226.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/08/26 09:31
-	244	compar$4 near10 list$3 near10 server$1 and @ad<20000803	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/12/10 09:25

PORTAL
THE ACM DIGITAL LIBRARY

> home : > about : > feedback : > login

US Patent & Trademark Office

Try the *new* Portal design

Give us your opinion after using it.

Search Results

Nothing Found

Your search for the *Phrase* **prox* <near/10> wildcard* <near/10> server*** did not return any results.
To search for *terms* separate them with **AND** or **OR**.
Click on the suggested options:

prox* AND <near/10> AND wildcard* AND <near/10> AND server*

prox* OR OR wildcard* OR OR server*

To search for names try using only the last or first name.

You may revise it and try your search again below or click advanced search for more options.

prox* <near/10> wildcard* <near/10> server*

SEARCH **[Advanced Search]** **[Search Help/Tips]**

Complete Search Help and Tips

The following characters have specialized meaning:

Special Characters	Description
, ( ) [	These characters end a text token.
= > < !	These characters end a text token because they signify the start of a field operator. (! is special: != ends a token.)
` @ \Q < { [!	These characters signify the start of a delimited token. These are terminated by the end character associated with the start character.

IEEE HOME | SEARCH IEEE | SHOP | WEB ACCOUNT | CONTACT IEEE

IEEE

Membership Publications/Services Standards Conferences Careers/Jobs

IEEE Xplore® RELEASE 1.5

Welcome
United States Patent and Trademark Office

Help FAQ Terms IEEE Peer Review

Quick Links

» Se

Welcome to IEEE Xplore®

- Home
- What Can I Access?
- Log-out

Tables of Contents

- Journals & Magazines
- Conference Proceedings
- Standards

Search

- By Author
- Basic
- Advanced

Member Services

- Join IEEE
- Establish IEEE Web Account
- Access the IEEE Member Digital Library

Print Format

Your search matched **[0]** of **[989552]** documents.

You may refine your search by editing the current search expression or entering a new one the text box. Then click search Again.

prox* <near/10> wildcard* <near/10> server*

Search Again

OR

Use your browser's back button to return to your original search page.

Results:

No documents matched your query.

Home | Log-out | Journals | Conference Proceedings | Standards | Search by Author | Basic Search | Advanced Search
Join IEEE | Web Account | New this week | OPAC Linking Information | Your Feedback | Technical Support | Email Alerting
No Robots Please | Release Notes | IEEE Online Publications | Help | FAQ| Terms | Back to Top

US006311216B1

(12) **United States Patent**
Smith et al.

(10) **Patent No.: US 6,311,216 B1**
(45) **Date of Patent: Oct. 30, 2001**

(54) **METHOD, COMPUTER PROGRAM PRODUCT, AND SYSTEM FOR CLIENT-SIDE DETERMINISTIC ROUTING AND URL LOOKUP INTO A DISTRIBUTED CACHE OF URLS**

(75) Inventors: **Brian J. Smith**, Seattle, WA (US); **Hans Hurvig**, Copenhagen (DK)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/086,843**

(22) Filed: **May 29, 1998**

(51) **Int. Cl.**[7] **G06F 15/173**; G06F 15/16

(52) **U.S. Cl.** **709/226**; 709/225; 709/229

(58) **Field of Search** 711/118, 122, 711/113, 14; 709/217, 218, 219, 226, 225, 229, 203, 105

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,341,499	*	8/1994	Doragh	709/301
5,539,883	*	7/1996	Allon et al.	709/105
5,603,029	*	2/1997	Aman et al.	709/105
5,612,865		3/1997	Dasgupta	364/184
5,623,595		4/1997	Bailey	395/182.04
5,649,093		7/1997	Hanko et al.	395/182.04

(List continued on next page.)

OTHER PUBLICATIONS

Microsoft Corporation. Cache Array Routing Protocol and Microsoft Proxy Server 2.0. World Wide Web: http://www-.microsoft.com. pp. 1–15. 1997.*

Sharp. Super Proxy Script. World Wide Web: http://narag-w.sharp.co.jp/sps/sps–e.html. pp. 1–9. Aug. 1996.*

Valloppillil, Vinod and Coben, Josh. Hierarchial HTTP Routing Protocol. World Wide Web. pp. 1–7, Apr. 1997.*

Valloppillil, Vinod and Ross, Keith W. Cacbe Array Routing Protocol v1.0. World Wide Web. pp. 1–9. Feb. 1998.*

Briefing on Super Proxy Script, (last modified Aug. 1998), http://naragw.sbarp.co.jp/sps/sps–e.html, pp. 1–9.

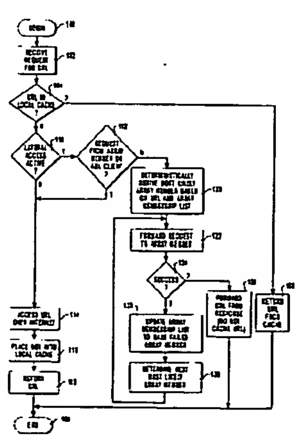
* cited by examiner

Primary Examiner—Glenton B. Burgess
Assistant Examiner—Nkosi Trim
(74) *Attorney, Agent, or Firm*—Workman, Nydegger & Seeley

(57) **ABSTRACT**

A method, computer program product, and system for directly accessing URL data object requests in a proxy server array. A URL data object request is generated by an enabled client to request a URL data object that resides in the local cache of proxy server in an array of proxy servers configured as a distributed cache. The enabled client will deterministically identify the residing proxy server based on information residing thereon without resorting to expensive query-response transactions, such as those that occur in proxy server arrays using ICP, or routing tbe URL data object request through different proxy servers of the array. An array membership list containing array membership information is available at each and every proxy server as well as all enabled clients. This list is used in conjunction with the URL as the information for identifying the correct proxy server where the URL data object resides. First, a deterministic hash value is computed for eacb proxy server name and the URL. Next, a combined hash value is computed that combines the URL bash value with each proxy server hash value. Finally, the proxy server with the highest "score" or combined bash value is identified as tbe proxy server where tbe desired URL data object sbould reside in local cacbe storage. Since the array membership list, the URL, and the hashing algoritbm are the same at all enabled clients, tbe same proxy server will be identified as having tbe URL data object regardless of which enabled client generated the URL data object request.

17 Claims, 11 Drawing Sheets



U.S. PATENT DOCUMENTS

5,740,371		4/1998	Wallis	395/200.59
5,774,660	*	6/1998	Brendel et al.	709/201
5,787,470		7/1998	DiSimone et al.	711/124
5,805,824		9/1998	Kappe	395/200
5,826,270		10/1998	Rutkowski et al.	707/10
5,864,852		1/1999	Luotonen	707/10
5,918,013		6/1999	Mighdoll et al.	395/200.47
5,924,116	*	7/1999	Aggarwal et al.	711/122
5,933,606		8/1999	Mayhew	395/200.69
5,933,849		8/1999	Srbljic et al.	711/118
5,935,207		8/1999	Logue et al.	709/219
5,940,594		8/1999	Ali et al.	395/200.33
5,987,233		11/1999	Humphrey	395/200.47
5,991,804		11/1999	Bolosky et al.	709/221
5,991,809		11/1999	Kriegsman	709/226
6,006,251		12/1999	Toyouchi et al.	709/203
6,006,264	*	12/1999	Colby et al.	709/226
6,014,667		1/2000	Jenkins et al.	707/10
6,026,405		2/2000	Arda et al.	707/10
6,029,168		2/2000	Frey	707/10
6,029,195		2/2000	Herz	709/219

* cited by examiner

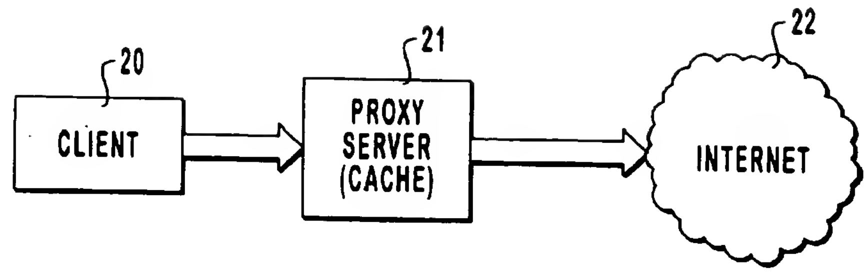


FIG. 1
(PRIOR ART)

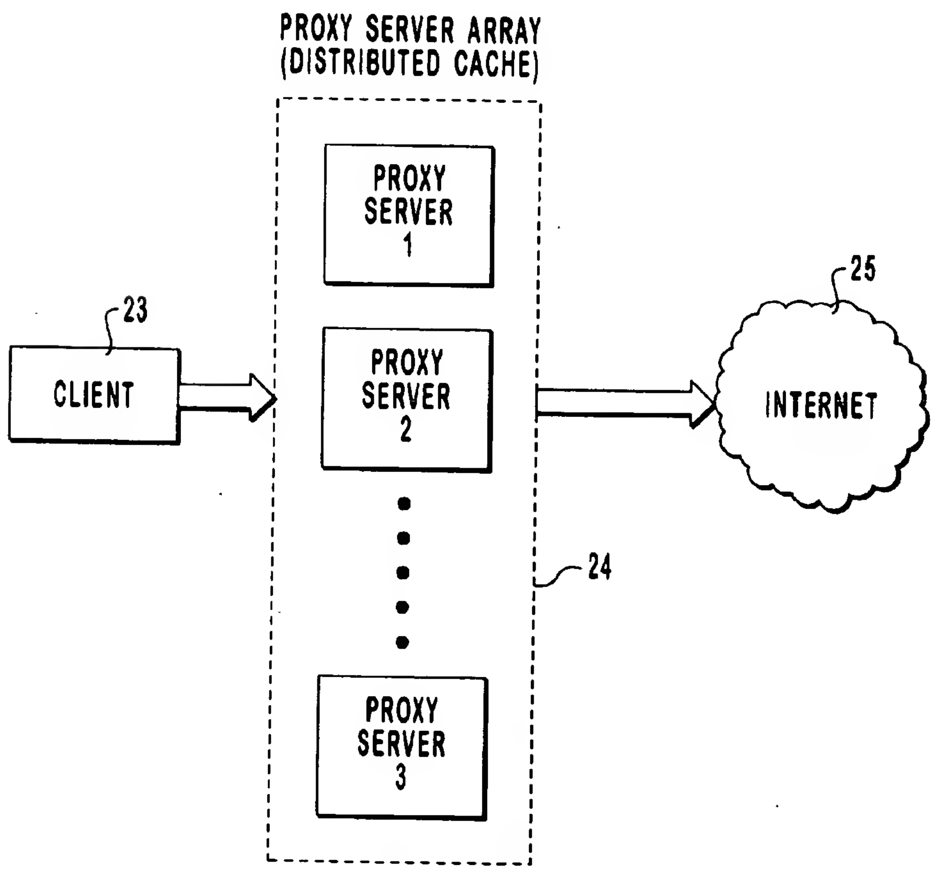


FIG. 2
(PRIOR ART)

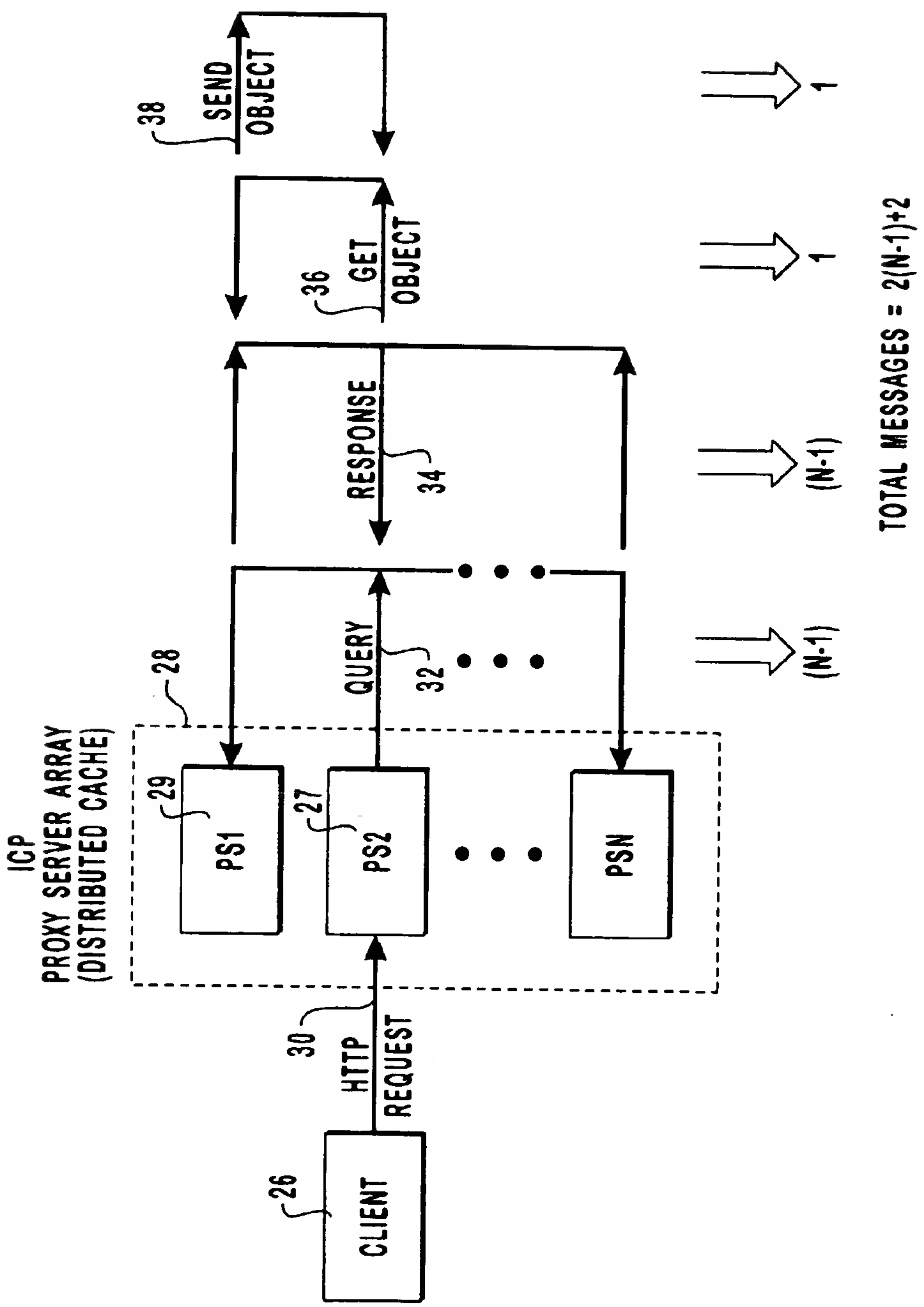


FIG. 3
(PRIOR ART)

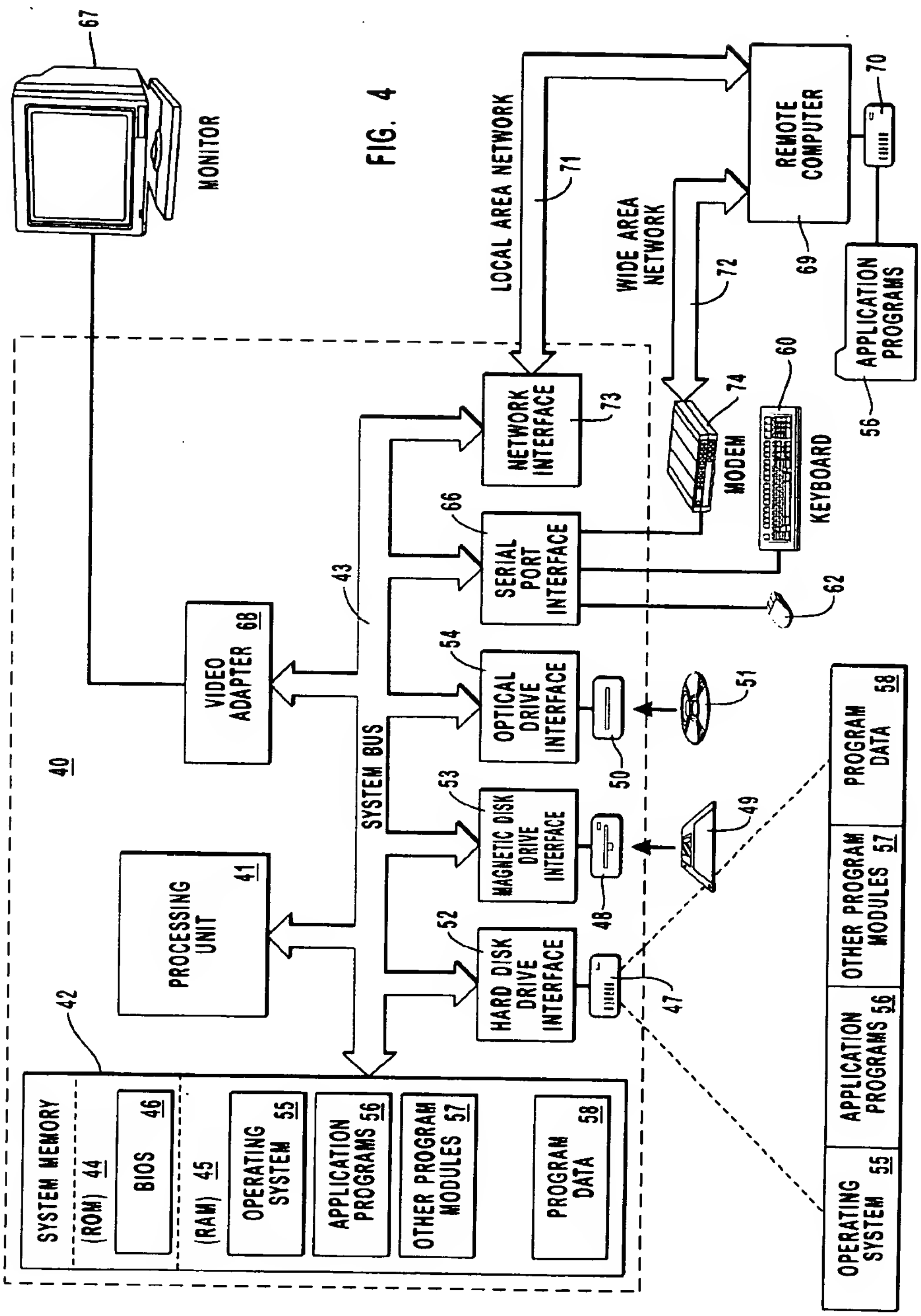

MONITOR
67
VIDEO ADAPTER 68
PROCESSING UNIT 41
SYSTEM BUS
43
40
42
SYSTEM MEMORY
(ROM) 44
BIOS 46
(RAM) 45
OPERATING SYSTEM 55
APPLICATION PROGRAMS 56
OTHER PROGRAM MODULES 57
PROGRAM DATA 58
HARD DISK DRIVE INTERFACE
52
MAGNETIC DISK DRIVE INTERFACE
53
OPTICAL DRIVE INTERFACE
54
SERIAL PORT INTERFACE
66
NETWORK INTERFACE
73
47
48
49
50
51
62
LOCAL AREA NETWORK
71
WIDE AREA NETWORK
72
MODEM
74
KEYBOARD
60
REMOTE COMPUTER
69
70
APPLICATION PROGRAMS
56
OPERATING SYSTEM 55
APPLICATION PROGRAMS 56
OTHER PROGRAM MODULES 57
PROGRAM DATA 58

FIG. 4

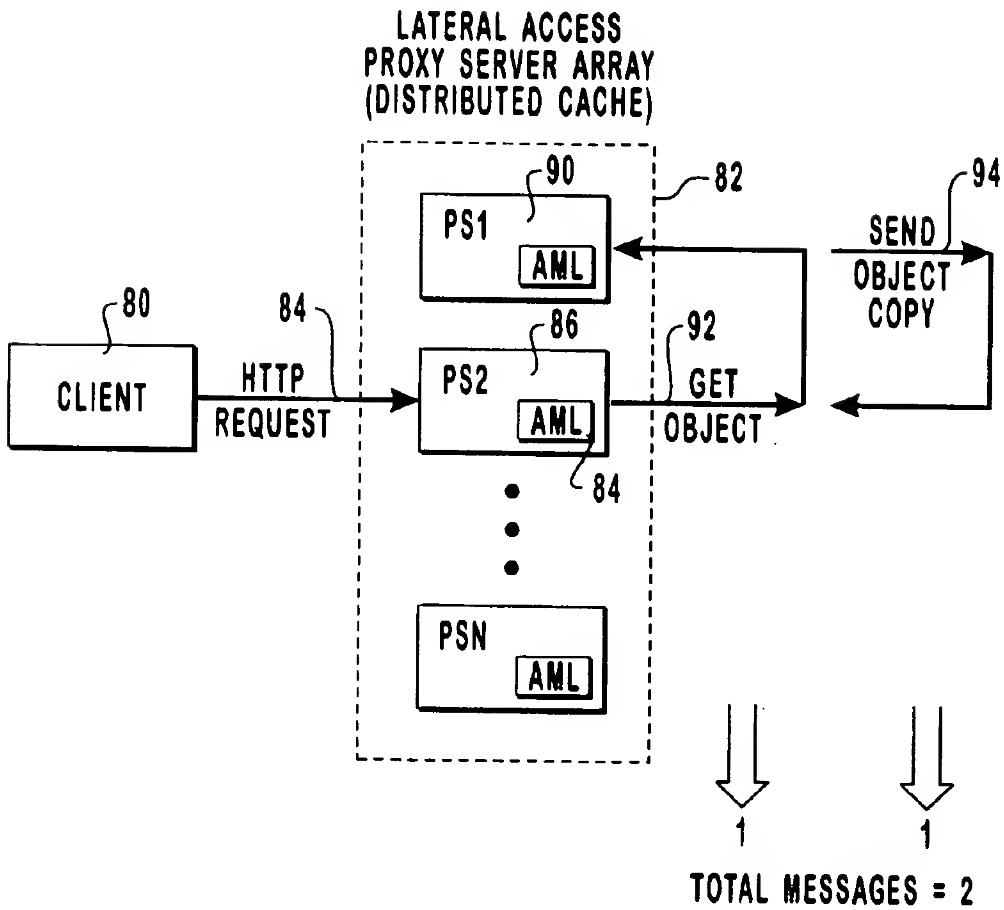

LATERAL ACCESS
PROXY SERVER ARRAY
(DISTRIBUTED CACHE)
90
82
94
PS1
AML
SEND
OBJECT
COPY
80
84
86
92
CLIENT
HTTP
REQUEST
PS2
AML
GET
OBJECT
84
PSN
AML
1
1
TOTAL MESSAGES = 2

FIG. 5

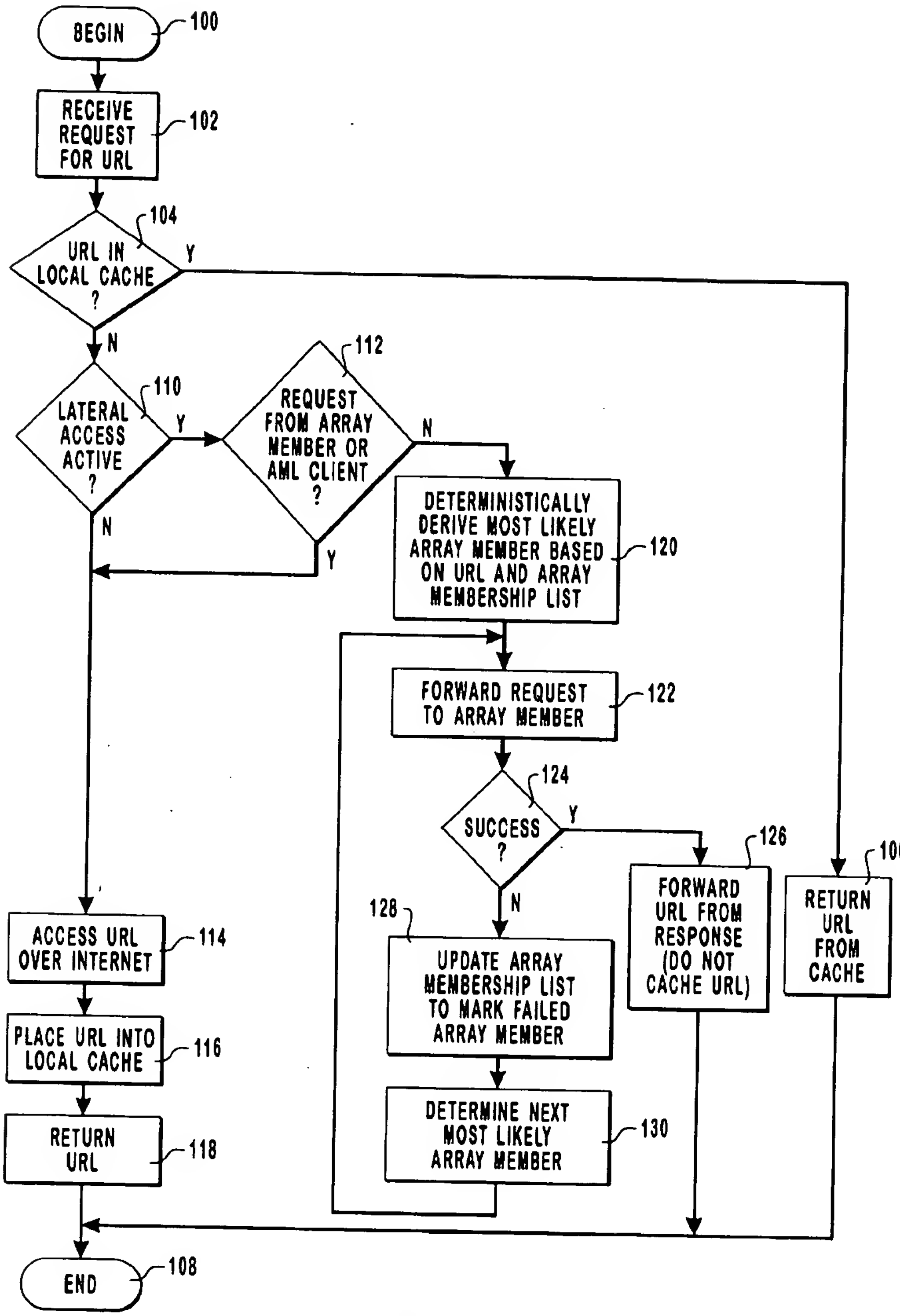

BEGIN
100
RECEIVE REQUEST FOR URL
102
URL IN LOCAL CACHE ?
104
Y
N
LATERAL ACCESS ACTIVE ?
110
Y
N
REQUEST FROM ARRAY MEMBER OR AML CLIENT ?
112
N
Y
DETERMINISTICALLY DERIVE MOST LIKELY ARRAY MEMBER BASED ON URL AND ARRAY MEMBERSHIP LIST
120
FORWARD REQUEST TO ARRAY MEMBER
122
SUCCESS ?
124
Y
N
FORWARD URL FROM RESPONSE (DO NOT CACHE URL)
126
RETURN URL FROM CACHE
106
128
UPDATE ARRAY MEMBERSHIP LIST TO MARK FAILED ARRAY MEMBER
DETERMINE NEXT MOST LIKELY ARRAY MEMBER
130
ACCESS URL OVER INTERNET
114
PLACE URL INTO LOCAL CACHE
116
RETURN URL
118
END
108

FIG. 6

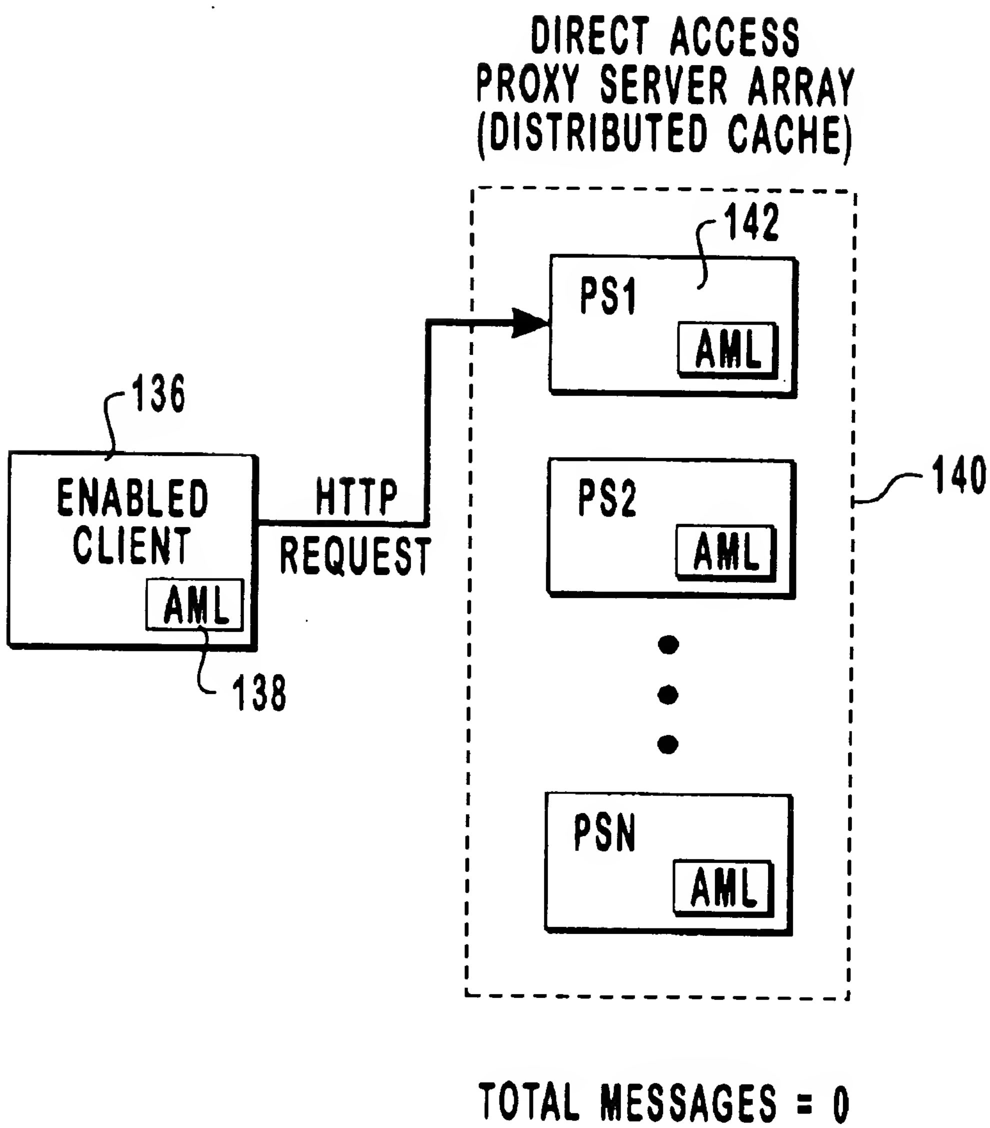

DIRECT ACCESS
PROXY SERVER ARRAY
(DISTRIBUTED CACHE)
142
PS1
AML
136
ENABLED
CLIENT
AML
HTTP
REQUEST
138
PS2
AML
140
PSN
AML
TOTAL MESSAGES = 0

FIG. 7

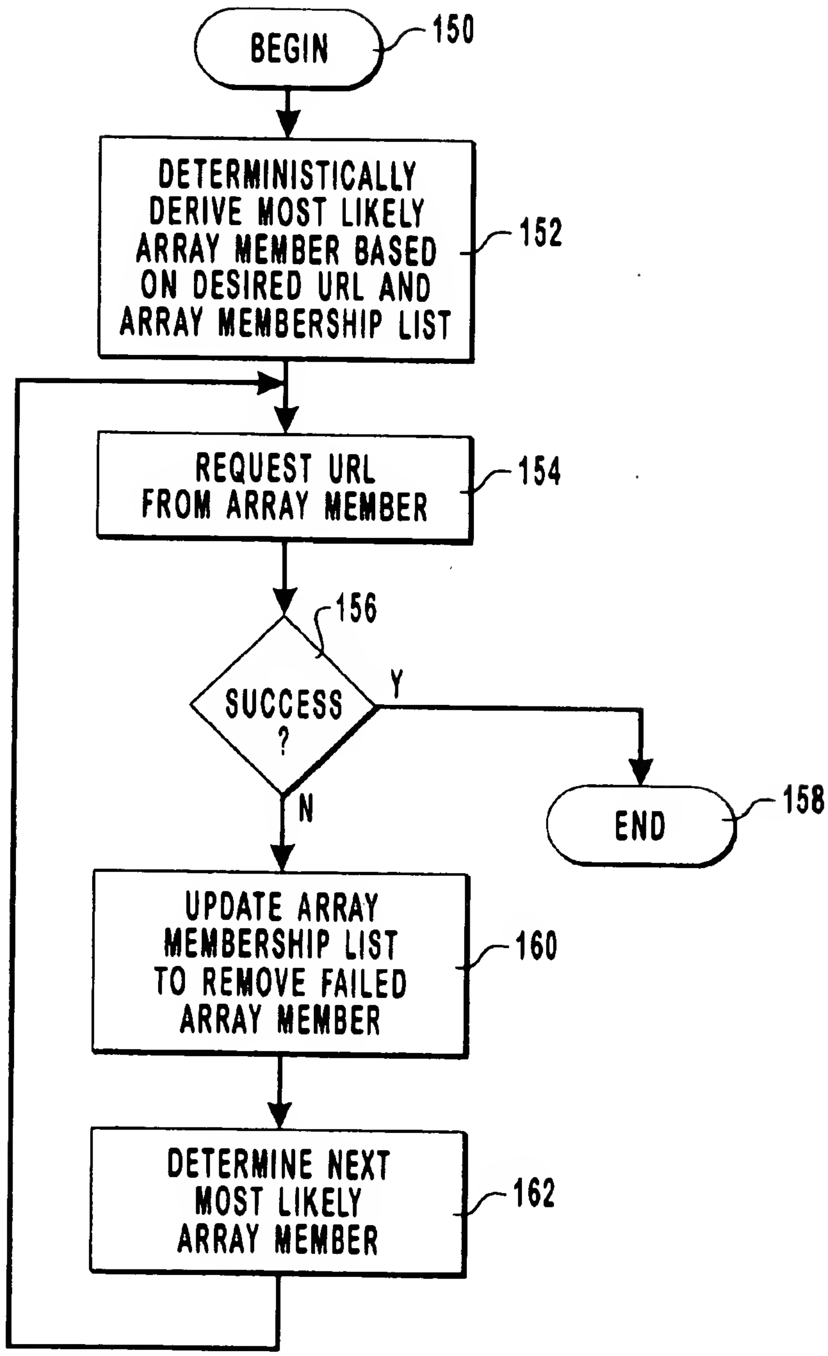

BEGIN
150
DETERMINISTICALLY DERIVE MOST LIKELY ARRAY MEMBER BASED ON DESIRED URL AND ARRAY MEMBERSHIP LIST
152
REQUEST URL FROM ARRAY MEMBER
154
156
SUCCESS ?
Y
N
END
158
UPDATE ARRAY MEMBERSHIP LIST TO REMOVE FAILED ARRAY MEMBER
160
DETERMINE NEXT MOST LIKELY ARRAY MEMBER
162

FIG. 8

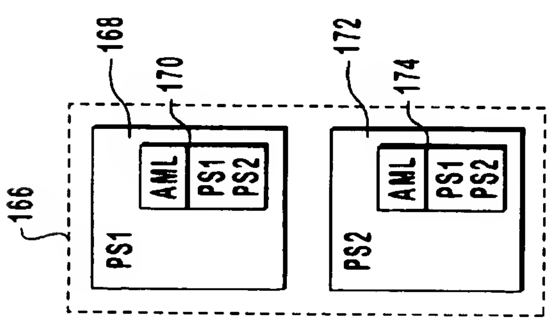


FIG. 9A

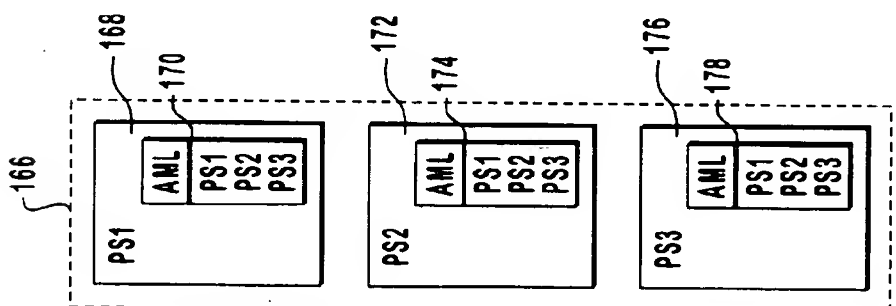


FIG. 9B

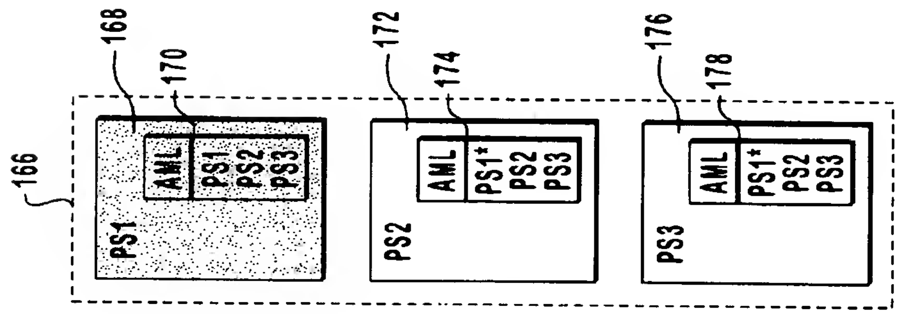


FIG. 9C

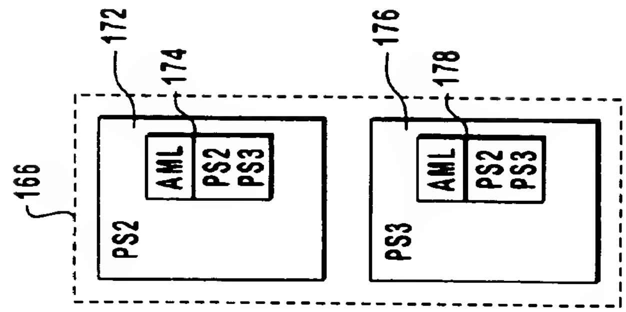


FIG. 9D

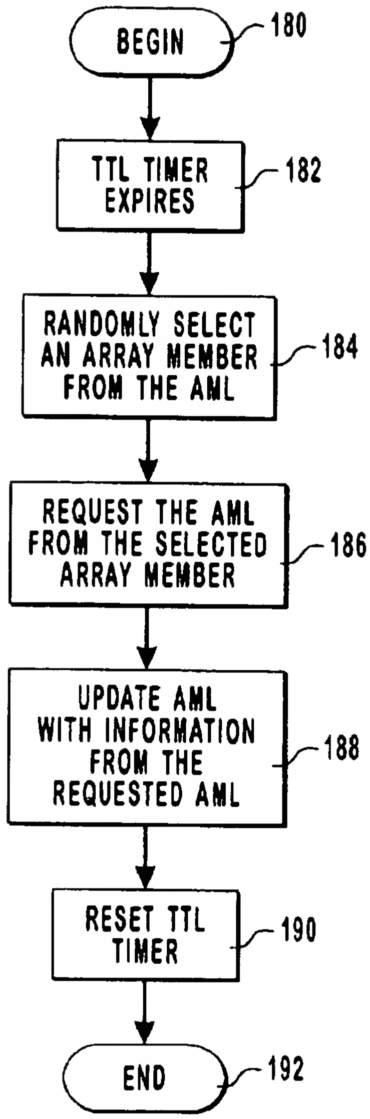

BEGIN
180
TTL TIMER EXPIRES
182
RANDOMLY SELECT AN ARRAY MEMBER FROM THE AML
184
REQUEST THE AML FROM THE SELECTED ARRAY MEMBER
186
UPDATE AML WITH INFORMATION FROM THE REQUESTED AML
188
RESET TTL TIMER
190
END
192

FIG. 10

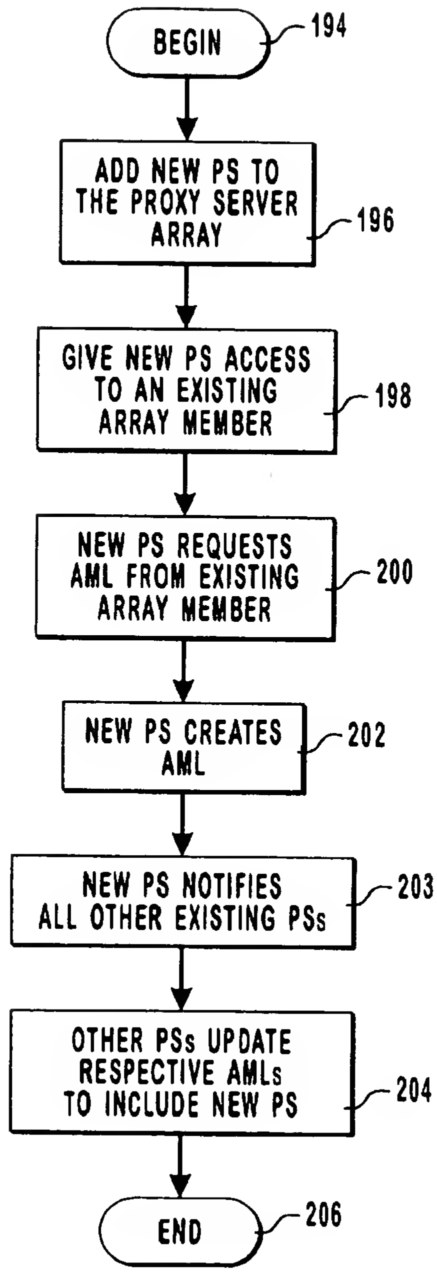

BEGIN
194
ADD NEW PS TO THE PROXY SERVER ARRAY
196
GIVE NEW PS ACCESS TO AN EXISTING ARRAY MEMBER
198
NEW PS REQUESTS AML FROM EXISTING ARRAY MEMBER
200
NEW PS CREATES AML
202
NEW PS NOTIFIES ALL OTHER EXISTING PSs
203
OTHER PSs UPDATE RESPECTIVE AMLs TO INCLUDE NEW PS
204
END
206

FIG. 11

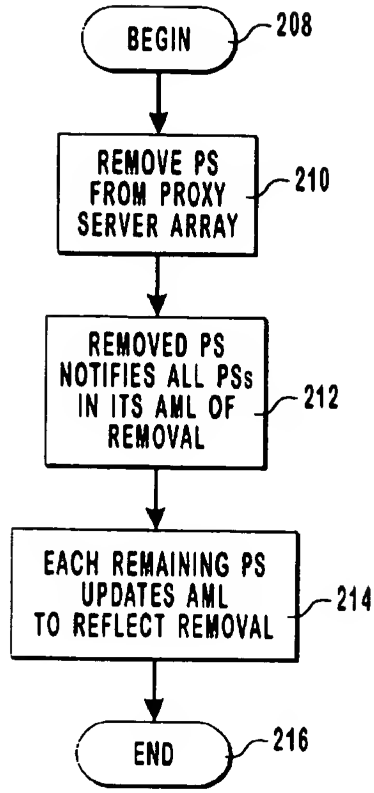

BEGIN
208
REMOVE PS FROM PROXY SERVER ARRAY
210
REMOVED PS NOTIFIES ALL PSs IN ITS AML OF REMOVAL
212
EACH REMAINING PS UPDATES AML TO REFLECT REMOVAL
214
END
216

FIG. 12

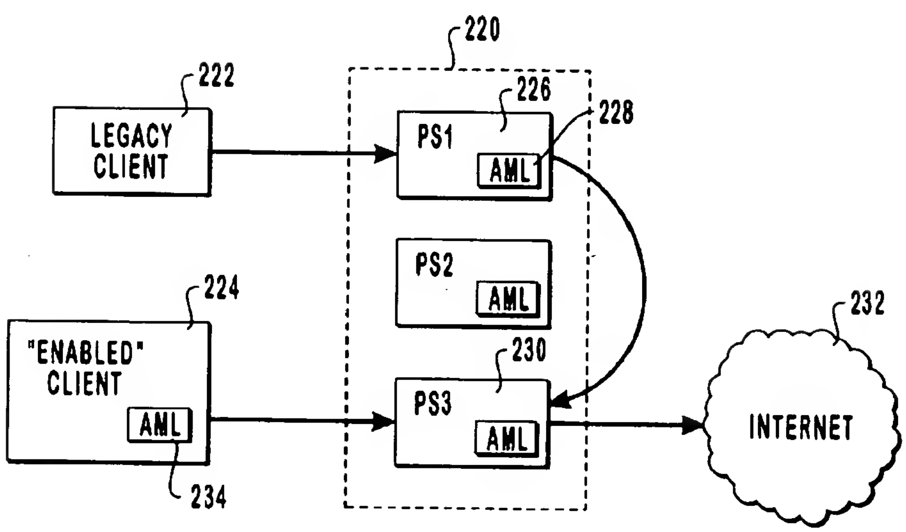

220
222
LEGACY CLIENT
226
228
PS1
AML
PS2
AML
224
"ENABLED" CLIENT
AML
234
230
PS3
AML
232
INTERNET

FIG. 13

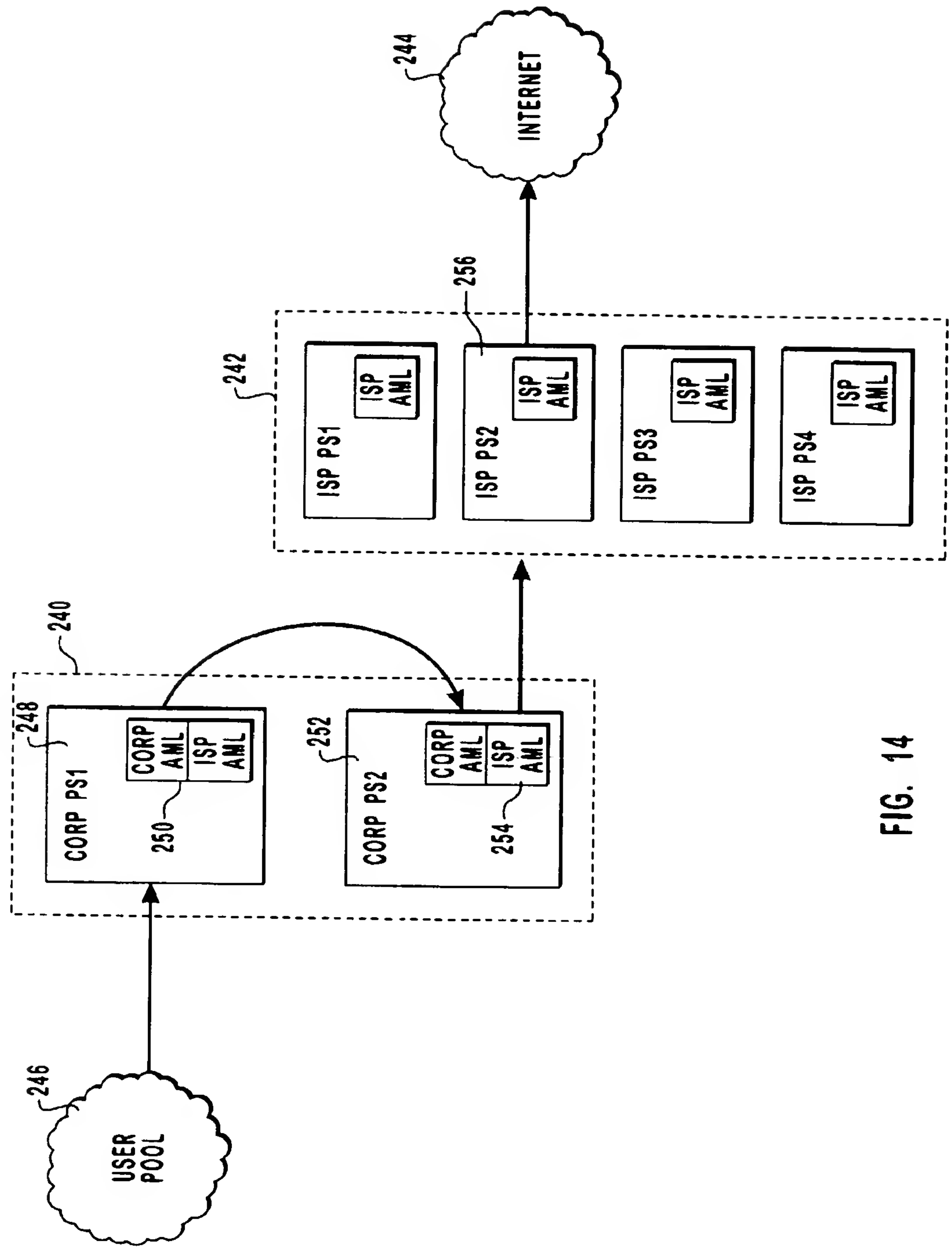

INTERNET
244
256
242
ISP PS1
ISP AML
ISP PS2
ISP AML
ISP PS3
ISP AML
ISP PS4
ISP AML
240
248
CORP PS1
CORP AML
ISP AML
250
252
CORP PS2
CORP AML
ISP AML
254
246
USER POOL

FIG. 14

METHOD, COMPUTER PROGRAM PRODUCT, AND SYSTEM FOR CLIENT-SIDE DETERMINISTIC ROUTING AND URL LOOKUP INTO A DISTRIBUTED CACHE OF URLS

BACKGROUND OF THE INVENTION

1. The Field of the Invention

The field of the present invention is that of the proxy servers used in connection with accessing data over the World Wide Web ("WWW") through the Internet or other Wide Area Network ("WAN"). More particularly, the present invention involves an array of multiple proxy servers configured together to act as a single distributed cache of information identified through the use of Uniform Resource Locators ("URL"). Specifically, the invention treats intelligent or enabled clients that may directly access a desired URL data object from a particular proxy server in a proxy server array having the URL requests laterally routed or transferred amongst array members.

2. Present State of the Art

Generally speaking, the concept of a "cache" or "caching" as used in computer terminology and applications typically means making a more accessible copy of some piece of data for a performance advantage. For example, information that is cached is in many instances more accessible than it otherwise would be so that processing speed is increased since accessing the cached information is quicker.

Since having excess copies of data can create its own sort of overhead and because cache size is limited, data is not typically retained in a cache indefinitely and will eventually be overwritten after a certain amount of time if the cache is being fully utilized. This may occur according to a Leased Recently Used ("LRU") algorithm, an expiration time, or any other relevant criteria for a particular application. Caching commonly exists at the microprocessor level with instruction and data caches so as to avoid excessive accessing of system RAM and may also exist elsewhere in a computer system or in a network of computer systems.

Another form of caching is commonly used in relation to accessing data or information over the WWW. A user having Internet access can directly receive a URL identified data object, such as a web page, and then display it locally on a browser. Each time the user receives such a data object, there is a delay as the Hyper Text Transport Protocol ("HTTP") request travels across the Internet to the location identified by the URL and the destination server processing the URL request responds with the requested data object. Because the data object can be quite large, the time to access the data objects with the HTTP request and response creates a significant amount of information traveling over the Internet connection causing delays and excess overhead.

In some organizations, many of the same data objects are requested by the various users within the organization. It is therefore common to introduce a proxy server that will receive user access requests from a client application, such as a web browser. Referring to FIG. 1, the use of a proxy server acting as a cache is shown. A client application **20** will direct all URL requests to a proxy server **21** that will serve as a cache for any information (i.e., URL data objects) associated with the URL. Should the proxy server **21** not have the URL data object within its cache, it will, in turn, make access over the Internet **22** to the destination server found within the URL in order to place the data object into its cache and then respond to the client **20** URL request. Thereafter, should another client make a request to the proxy server **21**, it can be serviced directly from the cache without necessary access over the Internet to the destination server found in the URL itself.

Note that the client **20** can be any software application capable of communicating or directing requests to the proxy server **21** using the HTTP protocol and would include other proxy servers, web browsers, Internet "enabled" applications, etc. Furthermore, HTTP requests contain a variety of information that can be used to "force" the proxy server **21** to access the URL data object over the Internet in order to assure that the "freshest" copy has been accessed. Such operation of proxy servers and their use as caches are generally known in the art for use in servicing HTTP requests.

It should be noted that the term URL may indicate either the address of where a data object is originally located and accessed, or the data object itself. To distinguish, a URL itself would be an address or location of the data object whereas a URL data object would be the actual web page file that is transmitted across the Internet to the client application. Though the appropriate usage is readily identifiable by context, efforts will be made throughout this application to distinguish between the two.

There are benefits of having a proxy server acting as a cache for URL data objects. One benefit is that for cached items, the total access time for a user is generally reduced since the connection between the client **20** and the proxy server **21** is typically over a Local Area Network ("LAN") rather than having to access the data object over the Internet or other Wide Area Network ("WAN"). Another benefit is for security purposes so that an organization may have a "firewall" to protect itself from unwanted outside penetration.

Larger corporations and other organizations may have many proxy servers servicing their needs. It becomes desirable in such situations to harness many proxy servers together as a single, logical distributed cache. Ideally, such a single distributed cache would have no duplication of URL data objects contained therein. Furthermore, a single distributed cache should have as little overhead as possible in servicing any given URL request that arrives at a member of the distributed cache. In other words, the actual URL data object may not be residing at the same server that originally receives the URL request and some form of forwarding, routing, or acquisition of the desired URL data object must occur in order to service that original URL request.

One attempt at creating a such a distributed cache is the Internet Cache Protocol ("ICP") that coordinates the activity of an "array" of proxy servers. Though ICP allows an array of proxy servers to function as a distributed cache, it also has some drawbacks as will be explained hereafter.

Referring now to FIG. 2, the interaction of a client with a proxy server array is shown. In such an arrangement, a client **23** will contact one of the proxy servers in the array **24** in order to access URL data objects that are available over the Internet **25**. Typically, a client **23** is assigned to a particular proxy server within the proxy server array **24** and may itself be a proxy server. Since the URL data object requested by a client may exist in a different proxy server than the one contacted, a mechanism or protocol is necessary for routing the URL request from the receiving proxy server to the appropriate proxy server, or in some other way service that URL request.

Referring now to FIG. 3, proxy server array that is organized and configured according to the ICP protocol is shown. In the example shown in FIG. 3, a client **26** will

direct HTTP requests to an assigned proxy server **27** that is part of an ICP proxy server array **28**. Assuming that a desired URL data object is contained in the distributed cache created by the ICP proxy server array **28** and located at the proxy server **29**, a scenario illustrating the operation of ICP is now shown. This scenario will also illustrate a number of problems that make ICP a less than optimal way of creating a distributed cache.

The URL request will originate at the client **26** and be received by the proxy server **27** as indicated by arrow **30**. After determining that the desired URL object does not reside at the proxy server **27** in its local cache storage, a query will be sent out to all proxy servers in the ICP proxy server array **28** as indicated by the query messages path **32**. In turn, every other proxy server that receives the query will give a response back to the proxy server **27** as indicated by the response message path **34**. Each individual response will indicate whether or not the specified URL data object resides at that particular responding proxy server. Note that more than one of the proxy servers in the ICP proxy server array **28** may contain a given URL data object.

For purposes of this example, it is assumed that only the proxy server **29** actually contains the data object and therefore the response from proxy server **29** to proxy server **27** would be the only response having an indication that the desired or requested URL data object exists thereon. Note that for an ICP proxy server array **28** of N proxy servers, that N−1 messages were sent out by the proxy server **27** querying for the existence of the URL data object and N−1 messages were sent back to or received by the proxy server **27** in response, thus creating a fair amount of network message traffic and usage of the overall network bandwidth.

Once the proxy server **27** knows where the desired URL data object is located, it will issue a request to get the object as indicated by the get path **36**. Naturally, the proxy server **29** will respond by sending the desired URL data object from the proxy server **29** to the proxy server **27** as indicated by the send object path **38**. Now that the proxy server **27** has the desired URL data object, it can respond to the original URL request and return that data object to the client **26**.

Cache storage redundancy can be seen since the proxy server **27** will also place the URL data object into its local cache such that the same URL data object now exists in both proxy server **27** and proxy server **29**. Also, network usage overhead is significant since the total number of messages to allow the proxy server **27** the ability to service the original URL request for an ICP proxy server array **28** of N proxy servers is 2(N−1)+2 total data messages across the network.

Given the above scenario, a number of undesirable problems exhibit themselves almost immediately. First, by using a query-response scenario to contact all of the proxy servers in the ICP proxy server array, a significant amount of network resources may be consumed. Additionally, a natural consequence of the query-response scenario is that the larger the ICP proxy server array **28** becomes the greater the network overhead for each non-resident URL request, therefore adding a negative scalability component to operating an ICP server array. This means that each addition of another proxy server onto the array will in fact increase the amount of communication between the different proxy servers for all array members and for each request in order to resolve the correct location of a desired URL data object. Theoretically, there may exist an upper limit to the number of proxy servers that may be comfortably used in an ICP proxy server array **28**.

Another problem is that multiple copies of the distributed cache URL data objects exist across the various proxy servers. In an extreme case, each proxy server of the ICP proxy server array **28** could become a redundant mirror of the other array members. It would be desirable to have only one copy of a URL data object existing in the entire distributed cache so that the distributed cache may be used to its full capacity.

Finally, adding or deleting proxy servers from the ICP proxy server array **28** may totally disrupt the distribution of URL data objects across the entire logical cache. In an extreme case, the distributed cache may be "emptied" upon any addition or removal of a proxy server from a proxy server array.

What is needed is a truly distributed logical cache across an array of proxy servers where a given URL data object is contained therein at only one location so as to maximize cache capacity. Furthermore, what is needed is a way to access the correct proxy server within the logical distributed cache or array of proxy servers without making a query to each and every proxy server making up the array thereby allowing a request received at one proxy server to be directly routed to the correct proxy server, or alternatively, an acquisition of the desired URL data object from the correct proxy server quickly attained. Finally, what is needed is a way of gracefully migrating cached URL data objects between the different proxy servers as a result of additions or removals of proxy servers from the proxy server array making up the distributed cache that will not require the reorientation of all URL data objects in the cache.

SUMMARY AND OBJECTS OF THE INVENTION

It is an object of the present invention to enable clients with the ability to directly access a proxy server in an array of proxy servers that will contain a desired URL data object based on processing the URL and proxy array membership information and without making expensive query-response transaction with each and every proxy array member.

It is another object of the present invention to utilize deterministic hashing algorithms to allow consistent and predictable identification of a proxy server to be assigned or have residing thereon a particular URL data object.

Additional objects and advantages of the invention will be set forth in the description which follows, and in part will be obvious from the description, or may be learned by the practice of the invention. The objects and advantages of the invention may be realized and obtained by means of the instruments and combinations particularly pointed out in the appended claims.

To achieve the foregoing objects, and in accordance with the invention as embodied and broadly described herein a method, computer program product, and system for client-side deterministic routing and URL lookup into a distributed cache of URLs are provided. An enabled client according to the present invention may directly access the correct proxy server in a proxy server array without making expensive query-response transactions or routing the URL data object request through multiple proxy servers.

Each proxy server has access to the entire array membership information stored in array membership list. This array membership list is periodically updated and reflects changes in array membership due to additions, removals, or temporary unavailability of the various proxy servers that make up the array. When changes have propagated through the proxy server array, all array membership lists at each proxy server will contain identical information.

The array membership list is also propagated to or otherwise made available to enabled clients so that array

membership information resides at the client itself. Many different mechanisms exist for getting the information to the enabled client as those skilled in the art may appreciate.

When an enabled client generates a URL data object request, it uses the array membership list, the URL itself, and a deterministic hashing function to identify which array member should actually hold the URL data object in its local cache. The request is then serviced by directly accessing the desired data object from the correct proxy server without making expensive query-response transactions over the network or routing the request through multiple proxy servers. The hashing function operates so as to distribute the cached URL data objects evenly over the entire proxy server array without redundancy so as to more efficiently use the array capacity.

The proxy server identification mechanism works by computing a hash value for each server name found in the array membership list and a hash value for the requested URL. The URL hash value is combined with each member proxy server hash value to form a combined value. The proxy server associated with highest of the combined values is identified as where the desired URL should reside in local cache. A load factor that assigns some proxy servers proportionately more URL data objects for the local cache is also incorporated in the creation of the combined hash values.

All information for making a determination as to the correct proxy server is completely available at an enabled client that generates the URL request so that no external information is necessary. Furthermore, because of the propagation of array membership information between the proxy servers, and from there to all enabled clients, the exact same information is used to identify the exact same proxy server regardless of which enabled client generates the URL request.

These and other objects and features of the present invention will become more fully apparent from the following description and appended claims, or may be learned by the practice of the invention as set forth hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

In order that the manner in which the above-recited and other advantages and objects of the invention are obtained, a more particular description of the invention briefly described above will be rendered by reference to specific embodiments thereof which are illustrated in the appended drawings. Understanding that these drawings depict only typical embodiments of the invention and are not therefore to be considered limiting of its scope, the invention will be described and explained with additional specificity and detail through the use of the accompanying drawings in which:

FIG. 1 is a logical diagram showing the use of a proxy server that serves as a cache from which a client may access desired URL data objects.

FIG. 2 is a logical diagram illustrating how an array of proxy servers may be logically configured to be a single distributed cache having a greater capacity.

FIG. 3 is a logical diagram showing an array of proxy servers that coordinate using the Internet cache protocol or ICP with an illustration of how a request directed into the cache at one proxy server may query all other proxy servers in the cache and eventually receive the data object laterally from another proxy server in order to service the original request.

FIG. 4 is a block diagram of an exemplary system for implementing the present invention that includes a general purpose computing device in the form of a conventional personal computer.

FIG. 5 is a logical diagram showing the operation of a proxy server array configured to form a distributed cache where requests are serviced according to the present invention and an example is given where a request received by one proxy server of the array may laterally access the proxy server having the desired URL data object without making queries to all proxy servers in the array.

FIG. 6 is a flow chart showing the processing steps taken by a proxy server according to the present invention when receiving a URL data object request.

FIG. 7 is a logical diagram showing an array of proxy servers configured to form a distributed cache according to the present invention and an enabled client according to the present invention wherein the client may make the original URL data object request directly to the proxy server within the array that is most likely to contain the desired URL data object.

FIG. 8 is a flow chart showing the processing steps taken by an enabled client for directly assessing the proxy server in a proxy server array that is most likely to contain the desired URL data object.

FIGS. 9A–9D show a logical sequence of events that may occur in the life of a proxy server array according to the present invention.

FIG. 9A shows the initial state of the proxy server array having two proxy servers and each Array Membership List ("AML") showing the two active members.

In FIG. 9B, a third proxy server is added to the array and the array membership list for all three proxy servers is updated to include the newly added proxy server.

In FIG. 9C, one proxy server is temporarily unavailable due to mechanical failure or other event and the array membership list of the remaining proxy servers in the array have been marked to indicate that the designated proxy server is temporarily unavailable.

In FIG. 9D, the same proxy server as was unavailable in FIG. 9C is actually removed from the array and the array membership lists from the remaining proxy servers are updated to indicate the removal of the proxy server from the list.

FIG, 10 is a flow chart showing the processing steps taken by a proxy server or enabled client for randomly accessing an array membership list from another member of the proxy server array that represents one form of communicating array membership information between the different proxy servers and enabled clients.

FIG. 11 is a flow chart showing the steps taken in order to add a new proxy server to the proxy server array.

FIG. 12 is a flow chart showing the steps taken for removing a proxy server from the array.

FIG. 13 is a logical diagram showing how a proxy server array according to the present invention may be used with legacy clients and enabled clients.

FIG. 14 is a logical diagram showing two levels of proxy server arrays according to the present invention may be used in a realistic environment where both lateral access and direct access are accomplished. A first proxy array uses lateral access within the array in order to service non-enabled or legacy clients, but in turn may directly access the proxy server array of an Internet service provider.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

As used herein, the term "hashing function" refers to those functions known in the art that systematically covert

one multi-bit representation into another, usually smaller, single or multi-bit representation. A hashing function is said to be deterministic if it generates the same results from the same input.

As used herein, the term "data object" refers to any data that may be accessed from a distributed store and singularly identified. Examples of data objects would include, but not be limited to, files, data base records, graphic images, programming "objects," etc. One particular data object that will used throughout the application is a URL data object that is any resource that can be identified and accessed using a URL according to the HTTP protocol. Typically, this would be a Hyper Text Markup Language ("HTML") file that is located on a server of the World Wide Web and accessed using a URL. Those skilled in the art will appreciate that the invention as explained using URL data objects accessed over the Internet will apply to many other environments having a distributed store of data objects.

As used herein, the term "array membership information" refers to information regarding all the servers making up an array of servers that can be configured into a distributed cache. As used more particularly throughout with one embodiment of the present invention, this would be information regarding proxy servers in a proxy server array that is used to cache URL data objects. Note that such array membership information may be incorporated into a file or data structure that may be shared or updated between the different array members, such as an array membership list. Array membership information would necessarily include some form of identification of each array member that can be used to access that member, such as server name or IP address. Additionally, array membership may include but not need not be limited to the following: a capacity for a particular server, a load factor that indicates a relative amount of load the server can handle as compared with other members of the array, and any information regarding server characteristics, such as CPU power, physical location, administrator's name, etc.

Unless specified, a URL request may refer to an original request or a forwarded request. The term "generating" when referring to a URL request or other data object access s request could be an original request or a forwarded request received by a server and simply passed on to another server.

As used herein, the term "distributed logical cache" or "distributed cache" refers to the cache as a whole over the entire proxy server array while "local cache" will refer to the data stored at a particular proxy server.

As used herein, the term "URL" will be used to designate both a URL data object as well as a URL in the sense of the identifying title of the URL data object that indicates the location and object name that would be included in an HTTP request for a URL data object (also known as a URL request). An HTTP response would include the actual URL data object (also known as a URL response).

As commonly used throughout, a URL data object is always accessed from the proxy server cache, even if not there initially. Any request received that is not in the cache will be placed therein after the proxy server gets it from the destination server over the Internet. Therefore, a cache is logically viewed for purposes of this application as containing all possible URL data objects that may be desired and when speaking of requesting an object from the distributed cache, the object is presumed to be in the cache.

It may be noted that the concepts of a proxy server acting as a store or cache for URL data objects may be applied to many different applications. For example, a distributed store of data files over a network could be achieved by having a proxy server servicing clients directly with cached copies of ordinary files that are located elsewhere in a corporation's network server structure. Those skilled in the art will see the natural application of the concepts involving proxy servers for accessing Internet information to other information access scenarios.

A "storage means," or "storage system" is defined broadly to incorporate any type of device interfacable with a CPU that is used to memorize information and includes both long-term and short-term storage. Thus, storage means would include, though not be limited to, cache memory, RAM, disk storage, tape storage, etc. Furthermore, storage means contemplates the entire system of storage incorporated by a computer in combination so that the RAM, cache, and disk drive together could be considered a storage means or storage system. A storage means can also be logically partitioned into different locations so that items are stored in different media or on different parts of the same media. For example, a storage means comprising RAM and disk storage could be logically partitioned so that item A is stored in a portion of RAM (first partition or location), item B is stored in another portion of RAM (second partition or location), and item C is stored on disk (third partition or location).

FIG. **4** and the following discussion are intended to provide a brief, general description of a suitable computing environment in which the invention may be implemented. Although not required, the invention will be described in the general context of computer-executable instructions, such as program modules, being executed by a personal computer. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

With reference to FIG. **4**, an exemplary system for implementing the invention includes a general purpose computing device in the form of a conventional personal computer **40**, including a processing unit **41**, a system memory **42**, and a system bus **43** that couples various system components including the system memory to the processing unit **41**. The system bus **43** may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) **44** and random access memory (RAM) **45**. A basic input/output system **46** (BIOS), containing the basic routines that helps to transfer information between elements within the personal computer **40**, such as during start-up, is stored in ROM **44**. The personal computer **40** further includes a hard disk drive **47** for reading from and writing to a hard disk, not shown, a magnetic disk drive **48** for reading from or writing to a removable magnetic disk **49**, and an optical disk drive **50** for reading from or writing to removable optical disk **51** such as a CD ROM or other optical media. The hard disk drive **47**, magnetic disk drive **48**, and optical disk drive **50** are connected to the system bus **43** by a hard disk drive interface **52**, a magnetic disk drive-interface **53**,

and an optical drive interface **54**, respectively. The drives and their associated computer-readable media provide non-volatile storage of computer readable instructions, data structures, program modules and other data for the personal computer **40**. Although the exemplary environment described herein employs a hard disk, a removable magnetic disk **49** and a removable optical disk **51**, it should be appreciated by those skilled in the art that other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROM), and the like, may also be used in the exemplary operating environment.

A number of program modules may be stored on the hard disk, magnetic disk **49**, optical disk **51**, ROM **44** or RAM **45**, including an operating system **55**, one or more application programs **56**, other program modules **57**, and program data **58**. A user may enter commands and information into the personal computer **40** through input devices such as a keyboard **60** and pointing device **62**. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit **41** through a serial port interface **66** that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port or a universal serial bus (USB). A monitor **67** or other type of display device is also connected to the system bus **43** via an interface, such as a video adapter **68**. In addition to the monitor, personal computers typically include other peripheral output devices (not shown), such as speakers and printers.

The personal computer **40** may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer **69**. The remote computer **69** may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the personal computer **40**, although only a memory storage device **70** has been illustrated in FIG. **2**. The logical connections depicted in FIG. **2** include a local area network (LAN) **71** and a wide area network (WAN) **72**. Such networking environments are commonplace in offices enterprise-wide computer networks, intranets and the Internet.

When used in a LAN networking environment, the personal computer **40** is connected to the local network **71** through a network or adapter **73**. When used in a WAN networking environment, the personal computer **40** typically includes a modem **74** or other means for establishing communications over the wide area network **72**, such as the Internet. The modem **74**, which may be internal or external, is connected to the system bus **43** via the serial port interface **66**. In a networked environment, program modules depicted relative to the personal computer **40**, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Referring now to FIG. **5**, a logical diagram showing a proxy server array according to the present invention that allows lateral access of a URL data object without querying all proxy servers in the array as would occur using ICP is shown. A client **80** will access a proxy server array **82** configured to a distributed cache with an original HTTP request **84** directed to proxy server **86**. The proxy server **86** has an array membership list **84** that contains all the proxy servers that make up the proxy server array **82**.

Based on the information in the array membership list **84** and the HTTP request containing the URL for the URL data object, the proxy server **86** will determine which proxy server in the proxy server array **82** should contain the desired URL data object. One way that this can be done, and which will be shown in more detail hereafter, is by using a deterministic hashing function to hash the URL received in the HTTP request **84** and combining it with a deterministic hash of each server name in the array membership list **84** in order to generate a rank ordering of each server in the array membership list **84** based on combined hash value. Typically, the combining of the URL hash value and the array membership list server hash values is itself another deterministic hash. Furthermore, once the proxy servers found in the array membership list **84** are rank ordered, the proxy server with the highest combined hash value or score is chosen for locally caching the URL data object. The deterministic hashing algorithms used will calculate hash values such that URL data objects are evenly distributed over the proxy servers making up the array. Again, choosing the correct proxy server within the proxy server array **82** that will contain the desired URL data object in the local cache will be explained in greater detail hereafter.

Once the correct proxy server is determined, proxy server **86** will direct a get object request to proxy server **90** as shown by a get object message **92**. Proxy server **90** will respond by sending a copy of the URL data object back to proxy server **86** as indicated by the send object message **94**. At this point, proxy server **86** may respond to the original HTTP request **84** thereby allowing the client **80** to have the requested URL data object.

It is important to note that regardless of which proxy server in the proxy server array **82** receives the original URL request, for a particular URL, the same proxy server will be indicated as having the URL data object in its local cache. This results from using deterministic hashing algorithms on exactly the same information since every proxy server in the proxy server array **82** will have an array membership list. This is one way to determine the correct proxy server within a proxy server array that will have a desired URL object without making queries between the proxy servers.

Only two messages are used to allow the proxy server **86** the ability to service the original URL request **84**. Such a proxy server array **82** does not exhibit the same negative scaleability problems associated with an ICP proxy server array since there is no need to have a query-response transaction between the proxy server receiving the original URL request and each and every other proxy server in the array. In fact, there is a positive scaleability since as more servers are added to the proxy server array **82**, the greater the capacity for the distributed cache.

An additional benefit of a proxy server array as shown in FIG. **5** over an ICP array is that the capacity of the distributed cache is used more effectively with fewer duplicate URL data objects stored at different local caches within the entire distributed cache. This occurs since the deterministic hashing mechanism determines one designated proxy server for each URL.

Referring now to FIG. **6**, a flow chart showing the processing steps taken by a proxy server found in the proxy server array **82** as shown in FIG. **5** illustrates the processing steps taken upon receiving a URL data object request. After beginning at step **100**, a proxy server, such as proxy server **86** in the proxy server array **82** of FIG. **5**, will receive a request for a particular URL data object at step **102**.

An initial check is made to determine whether or not the URL is in the local cache at step **104**. If the URL data object

is in the local cache, the response to the originally received URL data object request will be serviced by returning the URL from the local cache at step 106 before processing ends at step 108. Naturally, taking the path of having the URL in local cache would be the quickest way to service a URL request.

As will be shown in greater detail hereafter, an "enabled" client having the array membership information could do the computations at the client for determining the correct proxy server for a particular URL data object. Therefore, the original URL data object request can be made directly to the proxy server most likely to have the desired URL data object.

Should the URL data object of the received request not be located in local cache as determined at step 104, a check is made at step 110 to determine whether lateral access is currently active for the proxy server array 82. If lateral access is not active, then the proxy server may operate as a single proxy server rather than as in a distributed cache or array member. Alternatively, if all clients to the proxy server array are "enabled" clients, meaning that they have the ability to directly access the designated proxy server for a particular URL data object, then it can be assured that the designated proxy server is the correct recipient of the URL data object request.

At step 114, the proxy server would access the URL data object over the Internet and receive a copy at the proxy server, followed by placing the URL data object into the local cache at step 116, and finally returning the URL data object to the client at step 118 as part of a response to the original URL data object request. Specific mention is made at step 114 of accessing the URL data object over the Internet for simplicity sake since proxy server arrays may be clients to other proxy server arrays. Eventually, however, the URL data object request will be finally resolved through the Internet if no intervening proxy servers or proxy server arrays contain the object locally. Finally, processing would end at step 108 until another URL data object request is received by the proxy server.

If lateral access is active as determined at step 110, indicating that the proxy server is operating as part of a distributed cache in a proxy server array such as that shown in FIG. 5, the request is analyzed to determine whether it was made from another member of the proxy server array or an "enabled" client having access to the array membership list and thus able to directly access the proxy server within a proxy server array that is most likely to contain the desired URL data object in its local cache. If so, then the proxy server receiving the request is the correct proxy server but simply does not have the desired URL data object in the local cache as previously determined at step 104. In such a case, the desired URL data object must be accessed over the Internet at step 114 and placed in to the local cache at step 116. At that point, the URL data object may be returned at step 118 to the client in an appropriate response before processing ends at step 108.

If the request was not from another proxy server that was part of the same proxy server array or a client that had an array membership list and thus could make direct accesses as determined at step 112, then the correct array member proxy server must be determined at step 120. In other words, at step 120, a deterministic way must be used to process available information in order to avoid the detrimental query-response scenario carried out by ICP. The array membership information as found in the array membership list and the URL as found in the URL data object request are deterministically processed so that a proxy server that should have the URL data object contained therein as part of the local cache can be ascertained.

While those skilled in the art may determine a variety of ways to deterministically arrive at a single proxy member that can be consistently calculated regardless of which proxy member is making the calculation, the discussion that follows presents one such way that works very effectively. Note that the only real requirements for step 120 are that deterministic results are calculated when given the same starting information, such as the URL name and the different servers that make up the proxy server array, and that queries need not be sent out to all array members. While the present embodiment has an array membership list that is available at each proxy server that makes up the array, other embodiments may get or have the array membership information differently than this embodiment yet still accomplish the purposes as explained.

Hashing algorithms can be used to deterministically categorize both the array members and the URL itself. The present method comprises the following steps: (1) the array membership list is analyzed and hash values are computed on each of the proxy names found therein, (2) the URL name that is used to access the URL data object is run through a hashing algorithm and a hash value computed thereon, and (3) the two hashes are then combined taking into account a load factor that is assigned to each proxy server. The combined hash values will then give a numerical value for each proxy server that is unique for the URL sought, such that the correct proxy server is chosen by taking the highest "score" or hash value with the corresponding proxy server being selected as the most likely proxy server to contain the URL data object in its local cache.

The hashing algorithm used is statistically designed so that URL data objects are assigned equally over N proxy servers in the proxy server array. When a proxy server hash value is combined with a URL hash value, relative load factors can be taken into account so that proxy servers with enhanced capacity will be assigned a proportionately greater amount of the URL data objects.

When changes are made to the proxy server array membership, the URL data object assignments among the array members may also change. A URL data assignment is where a particular URL data object would reside if requested from the proxy server array. In other words, every possible URL data object will have an "assignment" to a particular proxy server in an array by operation of the deterministic hashing algorithm.

The assignment changes are automatically manifest by the operation of the hashing algorithm since all the proxy server making up the array will be used as input. Therefore, if an addition or deletion occurs, the algorithm will function differently, though still deterministically. For example, if an array is increased by one proxy server, each of the original proxy servers will be assigned fewer of all possible URL data objects to store in their local cache due to the addition of the new proxy server. In like manner, if a proxy server is removed from the array, each remaining proxy server will be assigned additional potential URL data objects from among all possible data objects to store in their respective local caches.

The assignment changes for the universe of URL data objects implies that, besides changes in routing of newly requested URL data objects, some actual URL data objects in one proxy server cache will be moved or migrated to another proxy server as a result of an addition or removal of

a proxy server from the array. Due to an array membership change, the same request that would previously indicate one particular proxy server will now indicate another proxy server as having the desired URL data object in the respective local cache. The URL data object would eventually be flushed from the first proxy server since no requests would be directed thereto due to the deterministic algorithm, while the new proxy server would receive the requests previously sent to the first proxy server. In this sense, the URL data object has "migrated" from one proxy server to another.

When speaking of URL data object migration throughout this application, it entails both the change in assignment for a given hypothetical URL data object that may be requested from amongst all possible URL data objects as well as actual URL data objects that shift between different proxy server local caches over time.

The deterministic algorithms used to make an "assignment" or route a URL data object request are designed so that only the minimum of URL data objects are switched between the proxy server in an array. For an array of N members, the addition of a proxy server will migrate 1/(N+1) of the total URL data objects taken equally from the original N proxy servers to the newly added proxy server while the rest will remain unchanged. In like manner, the removal of a proxy server will cause the 1/N of the total URL data objects to migrate equally among the remaining N−1 proxy servers.

Below is a more detailed example of how the determination is made at step **120**. Initially, the names of all the proxy servers in the array are taken and hash values computed for each, followed by the computation of a hash value of '19' for URL #1, and finally, a combined hash value for each proxy server as shown below in Table 1. Note that, if the proxy servers are ordered from highest to lowest combined hash value and the selection criteria for the designated proxy server is choosing the highest "score" or combined hash value, the proxy server "SweetHeart" is identified or chosen to cache the data object associated with URL #1 in its local cache.

TABLE 1

		URL #1
Proxy Server Name	Hash Value	19
PearBlossom	13	5
SweetHeart	8	9
Honey	5	7
Kitten	28	4

Below, in Table 2, scores are generated for URL #2, URL #3, and URL #4. Note that the "winning" scores highlighted below for the all the URLs indicate a natural load balancing with one URL data object being stored or assigned to each of the respective proxy servers.

TABLE 2

		URL #1	URL #2	URL #3	URL #4
Proxy Server Name	Hash Value	19	14	5	2
PearBlossom	13	5	6	10	4
SweetHeart	8	9	2	7	5
Honey	5	7	4	3	10
Kitten	28	4	7	8	1

The effect of adding an additional proxy server can be seen below in Table 3. A new proxy server named "Heidelburg" has been added to the proxy server array with its own combined hash values for each URL. Since the combined hash value for the proxy server named "Heidelburg" and URL #2 is the highest score, the data object associated with URL #2 will migrate from the proxy server named "Honey" to the proxy server named "Heidelburg."

TABLE 3

		URL #1	URL #2	URL #3	URL #4
Proxy Server Name	Hash Value	19	14	5	2
PearBlossom	13	5	6	10	4
SweetHeart	8	9	2	7	5
Honey	5	7	4	3	10
Kitten	28	4	7	8	1
Heidelburg	14	2	9	4	6

While those skilled in the art will appreciate that many different ways may be devised to generate the hash values used in the previous tables as long as they are deterministic in nature. One form of deterministic hash algorithms used to create hash values for the proxy servers making up an array, the URL, and the combined hash values is explained below.

Because irreversibility and strong cryptographic features are unnecessary for this application, a very simple and fast hash function based on the bitwise left rotate operator is used on each textual character making up the URL. For each character in the URL or the proxy server name, one of the following respective functions is performed to arrive at the URL hash value:

URL_Hash_Value+=rotl(URL_Hash_Value, 19)+<character_value>

Proxy_Server Hash_Value+=rotl(Proxy_Server_Hash_Value, 19)+<character_value>

Additionally, the following steps are taken on the proxy server hash value in order to further spread the values across the hash space since proxy server names may be similar to each other:

Proxy_Server_Hash_Value+=Proxy_Server_Hash_Value *0x62531965

Proxy_Server_Hash_Value+=rotl(Proxy_Server_Hash_Value, 21)

The combined hash value is created by using the exclusive-OR function on the URL hash value and the proxy server hash value, multiplying by a constant, and performing a bitwise rotation according to the following formulas:

Combined_Hash_Value=(URL_Hash_ValueÔProxy_Server_Hash_Value)

Combined_Hash_Value+=Combined_Hash_Value* 0x62531965

Combined_Hash_Value=rotl(Combined_Hash_Value, 21)

While these particular hashing functions can be used, those skilled in the art may develop many others that will work within the framework of the present invention. A load factor value associated with each proxy server may also be applied to each combined hash value in order to create a higher score for those proxy servers that have more capacity than others in a proxy server array.

Once the proxy servers have been ordered by means of the combined hash and a "winning" proxy server selected, the URL data object request is forwarded to the highest non-failed array member at step **122**. A test is made at step **124** to determine whether forwarded request was successful, and if so, then the desired URL will be contained in the response to the forwarded request. That URL data object can then be returned in response to the original URL data object request made to the server at step **126**.

Since this is a lateral access for the URL data object, it will not be placed in the local cache. This is because it already exists in the local cache of the proxy server that was successfully accessed and to place it in this proxy server's cache would create duplicate URL data objects in the overall distributed cache, thereby decreasing overall cache efficiency and capacity. At this point, processing would end at step **108** since the original URL data object request had been serviced.

If the forwarded request was unsuccessful as determined at step **124**, meaning that a timeout occurred and no response was received, then the array membership list is updated at step **128** to indicate a particular proxy server has failed and is temporarily unavailable. This is likely to occur when a proxy server is taken off line due to mechanical failure, system maintenance, or other reason rendering it temporarily unavailable to service HTTP requests.

At step **130**, the next most likely array member is determined by taking the highest score of the proxy servers in the array that have not been marked as temporarily unavailable (e.g. failed a request). Once the next proxy server is determined step **130**, processing loops back to forwarding the request to that array member at step **122**, and subsequent processing will be as explained previously.

Referring now to FIG. **7**, an enabled client **136** will use its own generated URL request in combination with the array membership list **138** in order to direct the original HTTP URL data object request into the proxy server array **140** directly to the proxy server **142** that is likely to have the desired URL data object in its local cache. The URL generated by the enabled client **136** may originate at the client in response to a user request or other stimulus or enabled client **136** may have received the URL data object request from elsewhere and may be simply forwarding such request. In any case, the term, "generated" refers to having a URL data object request prepared so that it may be sent out to the proxy server array **140**. Note that an enabled client may be a proxy server that is part of a proxy server array and that the terms client and server are used to indicate relationships between computer systems. In other words, the same computer system may be a client or server depending on context.

By processing the information from the URL data object request and the array membership list **138**, the enabled client **136** is able to directly access the correct proxy server **142** within the proxy server array **140**. The determination of the correct proxy server, such as proxy server **142**, can be done in the same manner as explained previously in connection for a lateral access within a proxy server array. This is accomplished by making the array membership list available to the enabled client **136**, and using the hashing algorithm that are used with the proxy servers within the array as explained previously in connection with step **120** of FIG. **6**.

This provides even more efficient service to an enabled client since there are no messages traded between the proxy servers making up the proxy server array **140**. By combining the lateral access as shown in the logical diagram of FIG. **5** and the flow chart of FIG. **6**, with the direct access logical model shown in FIG. **7** and explained in connection with the flow chart of FIG. **8**, both enabled clients such as enabled client **136** of FIG. **7**, and legacy or existing clients such as client **80** of FIG. **5**, can coexist in the same environment. This sort of arrangement will be explained in more detail hereafter in the discussion surrounding FIG. **13**. By combining direct access by enabled clients and lateral access amongst array members, a proxy server array may provide backwards compatibility while allowing greater flexibility for more intelligent or enabled clients.

Referring now to FIG. **8**, a flow chart showing the processing steps taken by an enabled client, such as enabled client **136** of FIG. **7**, in order to directly access the correct proxy server containing the desired URL data object in a proxy server array is shown. After beginning at step **150**, a URL request is generated in some fashion as explained previously. At step **152**, the enabled client will deterministically derive the most likely array member based on the generated URL and the array membership list containing array membership information. One preferred way of doing this has been explained previously in connection with a proxy server in the discussion of step **120** of the flow chart shown in FIG. **6**. The same processing would occur in the same preferred fashion for an enabled client and the details will not be repeated here.

Once the most likely array member has been determined, the URL data object is requested from that array member at step **154**. Should the array member return the desired URL data object as determined at step **156**, processing will end at step **158** since the client has received the desired URL data object.

Should there be a timeout or other problem such that success was not achieved at step **156**, the array membership list will be updated to temporarily mark the failed array member as unavailable at step **160**. As explained previously, this may occur due to any reason the array member is temporarily not available and it is expected eventually to return on-line.

At step **162**, the next most likely array member is determined. Again, this is done by taking the proxy server having the highest combined hash value that has not been marked as unavailable. Once the proxy server is determined at step **162**, processing loops around so that the enabled client may request the URL data object from the newly determined array member at step **154**. Again, processing will continue as explained previously.

Referring now to FIGS. **9A** through **9D**, a single proxy server array is shown that undergoes a number of different changes due to the addition, removal, or temporary unavailability of a proxy server in the array. Initially, the proxy server array **166** exists as shown in FIG. **9A** with a first proxy server **168** having an array membership list **170** and a second proxy server **172** having an array membership list **174**. The array membership lists **170** and **174** contain relevant array membership information including a reference to each and every other proxy server found within the

array as shown in FIG. 9A. Note that such reference may be by title, such as ASCII text location and name of the proxy server, by machine readable designation of the proxy server, such as an IP address, or any other means that allnws a proxy server to be uniquely identified.

In FIG. 9B, a third proxy server 176 with an array membership list 178 has been added to the proxy server array 166 according to the steps shown in FIG. 11 as will be shown in more detail hereafter. Once the third proxy server 176 has been added to the proxy server array 166, array membership information will be communicated between first proxy server 168, second proxy server 172, and third proxy server 176 such that the corresponding array membership lists reflect the addition of the third proxy server 176 as will be explained later in connection with the steps shown in the flow chart of FIG. 10.

Referring now to FIG. 9C, situation that nccurs when a proxy server is temporarily unavailable is shown. Namely, the first proxy server 168 is shaded to indicate its temporary unavailability due to maintenance, malfunction, or other such event. Furthermore, the array membership lists 174 and 178 of the respective second proxy server 172 and third proxy server 176 have the reference to the first proxy server 168 "marked" in some fashion to indicate its unavailability.

The marking of a proxy server as temporarily unavailable occurs during routing of the URL data object request as part of a lateral access amongst proxy server in the array as explained in connection with the discussion of steps 124 and 128 of flow chart in FIG. 6 or as part of an enabled client's direct access into the array as explained in connection with steps 156 and 160 of the flow chart shown in FIG. 8. As soon as the first proxy server 168 again becomes available, knowledge of its availability will be made known throughout the proxy server array 166 according to the loosely coupled array membership information propagation mechanism explained in more detail hereafter in connection with the flow chart shown in FIG. 10.

Referring now to FIG. 9D, the removal of the first proxy server from the proxy server array is shown according to the steps explained in more detail hereafter in connection with the flow chart shown in FIG. 12. Namely, the first proxy server 168 is completely removed from the proxy server array 166 thereby leaving the second proxy server 172 and the third proxy server 176 as the only members left in the array. The data objects locally cached at the removed proxy server 168 will be redistributed or migrated equally (or equally and according to load factor) with the remaining two proxy servers. Again, the corresponding array membership lists 174 and 178 reflect the removal of the first proxy server 168.

Referring now to FIG. 10, a flow chart showing the steps taken for one method of communicating proxy server array membership information between proxy servers and to enabled clients is shown. This is done by sharing array membership lists between the proxy servers making up the array on a periodic basis so that a change made to one array membership list will eventually propagate over to the other proxy servers in the array within a relatively short period of time. Such a "loose" propagation mechanism works well within the proxy server array environment even though those skilled in the art that other methods, such as broadcasting changes to all array members may provide more immediate results.

The same general mechanism can be utilized by an enabled client to receive relevant changes in array membership information. It is important to note that the use of array membership lists and the presently explained method of communicating the array membership lists between array members constitute only one way of providing and communicating proxy server array membership information across the array and those skilled in the art will appreciate that many different ways and means may be implemented to accomplish having valid array membership information available at an enabled client or an array member so that a deterministic routing of a URL data object request may occur based nn that array membership information along with the URL data object request itself.

Essentially, a time-to-live ("TTL") timer is implemented that triggers the operatinn of the processing steps shown in flow chart 10 for either an enabled client or a member proxy server. After processing begins at step 180, the TTL timer expires at step 182 indicating initiation of an update to the current array membership list. An array member is randomly selected from amongst the "active" proxy servers found in the current array membership list (e.g., not "marked" as temporarily unavailable) to receive a request for that proxy server's array membership list.

At step 186 a request for the array membership list of the selected array member is made. Once a response is received containing the array membership list, the proxy server will update its own array membership list with information from the newly received membership list. This may occur by direct replacement or some form of differential analysis to bring the proxy servers array membership lists up to date based on the newly received information. Finally, the TTL timer is reset at step 190 before processing ends at step 192 to allow the process to repeat itself periodically as needed. Typically, having the TTL timer expire every second or so provides enough periodic resolution to effectively propagate array membership information in a timely manner.

Proxy servers that are marked as temporarily unavailable will automatically become "unmarked" or available through each cycle of the update process. In this manner, whenever a previously unavailable proxy server becomes available, it will be detected by an enabled client or a member of the proxy server array. Essentially, when a proxy server is discovered to be temporarily unavailable, the enabled client or member of the array will not attempt any more accesses to the unavailable proxy server for the duration of the TTL time period. All marking is temporary so that only complete array membership lists are exchanged and a proxy server that continues to be unavailable will be discovered anew during the normal request processing.

Referring now to FIG. 11, the steps taken in order to add a proxy server to the proxy server array are shown. Again, many different implementations may be envisioned by those skilled in the art that will allow a proxy server to be added to the proxy server array.

After beginning at step 194, a new proxy server is designated as being added to the array at step 196. In order to make this proxy server available to other members of the array and enabled clients, the new proxy server is given access or indication of at least one existing array member at step 198 so that the new proxy server can request an array membership list from that existing proxy server at step 200.

Upon receiving the request, the existing array member returns its array membership list and the new proxy server creates its own array membership list at step 202 using knowledge of the existing array member and the received array membership list. Next, at step 203, the new proxy server notifies all members in its newly created array membership list of its existence in the proxy server array. Finally, at step 204, each existing member of the array member will update its respective array memebership list in response to the notifications sent at step 203 before processing ends at step 206.

At this point, the newly added proxy server may now begin randomly accessing all members of the array for updating its own array membership list and routing URL data object requests to the correct members of the array. As explained previously, FIG. 9B shows the state of the array previously shown in FIG. 9A after the addition of the third proxy server **176** at a point in time when the array membership information has been propagated to all array membership lists indicating the addition of the third proxy server **176**.

Referring now to FIG. **12**, flow chart showing the steps for removing a proxy server from a proxy server array is now shown. After beginning at step **208**, a particular proxy server is designated as removed from the proxy server array at step **210**.

At step **212**, the designated proxy server notifies all proxy servers in its array membership list of its removal. The removed proxy server will no longer respond to requests directed thereto or may respond with some form of indication that it is no longer a member of the proxy server array until its removal has propagated to all array members and enabled clients. Each proxy server remaining in the array will receive notification from the removed proxy server at step **212** and will update its respective array membership list to reflect the removal at step **214** so that no more attempts at accessing the removed proxy server for either the array membership list or for routing of a URL data object request will be made. Finally, processing ends at step **216**.

Referring now to FIG. **13**, a logical diagram showing a proxy server array with lateral access between array members that will accommodate both legacy clients and enabled clients is presented. A legacy client **222** will access the proxy server array **220** in the traditional fashion by being assigned a specific proxy server while an enabled client **224** will utilize proxy server array member information in order to directly access the correct proxy server within the array having the desired URL data object as explained previously in connection with the logical diagram of FIG. **7** and the flow chart shown in FIG. **8**.

Assume the legacy client **222** directs a URL data object request to the first proxy server **226** of the proxy server array **220** by way of assignment. The first proxy server **226** determines that the desired URL data object does not reside in its local cache and uses the URL data object request and the proxy server array **220** membership information located in the array membership list **228** to ascertain that the desired URL data object should reside at the third proxy server **230**. At that point, the request will be forwarded from the first proxy server **226** to the third proxy server **230** as indicated by the arrow. This determination and forwarding of the URL data object request received from the legacy client **222** at the first proxy server **226** is done according to step **120** and step **122** of the flow chart of FIG. **6** as such processing steps are executed at the first proxy server **226**.

The third proxy server **230**, upon receiving the forwarded URL data object request, will, for purposes of this example, determine that the URL is not in the local cache at step **104** of FIG. **6**, determine that lateral access is active at step **110**, and that the request is from an array member at step **112**. This basically indicates that the desired URL data object does not exist in the distributed cache and must be accessed over the Internet **232** and placed into the third proxy server **230** local cache as was explained previously as step **114** and step **116** of FIG. **6**.

Finally, with the desired URL data object residing in its local cache, the third proxy server **230** may return the desired URL data object to the first proxy server **226**, which in turn will service the original request from the legacy client **222**. Note that the first proxy server **226** will not store an extra copy of the URL data object in its local cache.

Assuming now that the enabled client **224** desires to access the exact same URL data object, it will use the generated URL data object request and the array membership list **234** in order to make the determination that the third proxy server **230** contains the desired URL data object. This occurs as was explained previously in connection with the logical diagram shown in FIG. **7** and steps **152** and **154** of the flow chart shown in FIG. **8**.

The third proxy server, upon receiving the URL request at step **102** of FIG. **6**, will further determine that the desired URL data object is located in the local cache at step **104** and then directly service the URL request by returning the URL data object found in the local cache at step **106**. Therefore, as shown in FIG. **13**, both enabled clients such as enabled client **224**, and older clients, such as legacy client **222**, utilize the same logical cache from the proxy server array **220** that has the direct and lateral accessibility.

Referring now to FIG. **14**, a logical diagram is shown wherein there is two levels of proxy server array caching, one for an internal corporate proxy server array and a second for an Internet service provider proxy server array. In the scenario illustrated in FIG. **14**, a corporate proxy server array **240** is an intelligent client to the Internet service provider proxy array **242** which in turn has access to the Internet **244** in order to connect with all servers on the World Wide Web. A particular user pool **246** consisting of a group of legacy clients, such as legacy Internet browsers in a department or division, is assigned to direct requests to the first corporate proxy server **248** that is part of the corporate proxy server array **240**.

Each proxy server in the corporate proxy server array **240** will have two array membership lists. The corporate array membership list will be used for managing lateral accessing in the corporate proxy server array **240** while the Internet service provider array membership list will be used by the corporate proxy server array **240** so that it may act as an intelligent client to the Internet service provider proxy server array **240**. Furthermore, the Internet service provider array membership list will be used by the Internet service provider proxy server array **242** for managing lateral routing therein.

Assuming a URL data object request from the user pool **246** arriving at the first corporate proxy server **248**, a determination is made at the first corporate proxy server **248** using the receive URL data object request and the corporate array membership list **250** that the desired URL data object should reside on the second corporate proxy server **252**. The first corporate proxy server **248** will forward the original URL data object request to the second corporate proxy server **252** as indicated by the arrow.

Once the second corporate proxy server **252** determines that external access is required from the Internet or other upstream proxy server array, the Internet service provider array membership list **254** will be used in conjunction with the forwarded URL data object request information in order to determine exactly which proxy server in the Internet service provider proxy server array **242** will contain the desired URL data object in its local cache. Note that the second corporate proxy server **252** is acting as an enabled client in making this direct access into the Internet service provider proxy server array **242**.

The result of such determination by the second corporate proxy server **252** is another forwarded URL data object request from the second corporate proxy server **252** to the second Internet service provider proxy server **256** that will in turn bring the desired URL data object into its local cache after accessing it over the Internet **244**.

The response path will pass the desired URL data object from the second Internet service provider proxy server to the second corporate proxy server **252**. The desired URL data

object will also be stored in local cache of the secood corporate proxy server 252 so that there are now two cached copies, ooe in the corporate proxy server array 240 aod ooe in the Iotemet service provider proxy server array 242.

The second corporate proxy server 252 will respood to the first corporate proxy server array 248 with the desired URL data object. Upoo receiving the response, the first corporate proxy server 248 will respood to the origioal URL data object request without storing the desired URL data object io its local cache to fioalize the traosaction chain. Those skilled in the art will recognize that may differeot topologies of proxy server arrays may be introduced that can be effectively serviced by the lateral aod direct accessing methods disclosed herein.

The present ioveotioo may be embodied in other specific forms without departing from its spirit or esseotial characteristics. The described embodiments are to be considered io all respects ooly as illustrated and oot restrictive. The scope of the inveotion is, therefore, indicated by the appended claims rather than by the foregoing description. All chaoges which come withio the meaning and range of equivalency of the claims are to be embraced within their scope.

What is claimed and desired to be secured by United States Letters Patent is:

1. In a client system associated with ao array of servers configured so as to provide a distributed store of data objects, a method of transmitting a request for a data object to a single server of the array that is assigned to store the data object without sending queries to each server in the array to ascertain the location of the data object, the method comprising the acts of:

providing array membership information at the client system, the array membership ioformation includiog information ideotifyiog each server that is active in the array at a given time;

providing, at the clieot system, information ideotifying a data object that is to be accessed by the client system;

determining which single server of the array is assigned to store the data object by performiog the acts of:

performing a first deterministic fuoctioo on the informatioo ideotifyiog the data object;

for each server, performing a second deterministic function on the informatioo ideotifying the server;

combining the results of the first deterministic function with the results of the second determioistic functioo to generate a value for each server; and

based on the relative values for the servers, deterministically ideotifying which single server is assigned to store the data object, without sendiog a query to each server to ascertain the location of the data object; and

transmitting an access request for the data object to the identified siogle server.

2. The method of claim 1 wherein the data objects are Uniform Resource Locator ("URL") data objects, the servers are proxy servers, the store of data objects is a cache of URL data objects, and the access request is a URL data object access request.

3. The method of claim 1 whereio the first deterministic function and the second deterministic fuoction are determioistic hash functions.

4. The method of claim 3 wherein the act of deterministically identifyiog which single server is assigned to store the data object comprises the steps of:

ordering the servers based on the values for the servers; and

identifying the server having the highest value as being the server assigoed to store the data object.

5. The method of claim 1 wherein the array membership information is contained in an array membership list that is distributed between the servers in the array aod accessible to the clieot system.

6. The method of claim 5 whereio the step of providing array membership informatioo at the clieot system comprises the acts of:

periodically requesting a oew array membership list from a randomly chosen server that is listed in a current array membership list stored at the clieot system; aod

updatiog the curreot array membership list with the oew array membership list.

7. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 1.

8. Io a client system associated with an array of proxy servers configured so as to provide a cache of uniform resource locator ("URL") data objects distributed across the array of proxy servers, a method of transmitting a request for a URL data object to a single proxy server of the array that is assigned to store the URL data object, the method comprising the acts of:

providing an array membership list at the client system, the list including names of all proxy servers that is active in the array at a given time;

providing, at the client system, a URL identifying a URL data object that is to he accessed by the client system;

determiniog which single proxy server of the array is assigoed to store the URL data object by performing the acts of:

for each of the proxy servers that are active io the array, computing a first deterministic hash value of the oame of the proxy server;

computiog a second deterministic hash value of the URL;

combining, for each of the proxy servers active in the array, the first deterministic hash value aod the secood deterministic hash value to georate a deterministic combioed hash value associated with each of the proxy servers active in the array; and

based oo the relative values of the deterministic combined hash values, deterministically identifying which single proxy server is assigoed to store the URL data object; and

traosmitting a URL access request for the URL data object to the ideotified single proxy server.

9. The method of claim 8 wherein the act of deterministically identifying which single proxy server is assigned to store the URL data object comprises the acts of:

ordering the proxy servers based on the values for the servers; and

ideotifying the proxy server having the highest deterministic combined hash value as being the single proxy server assigned to store the data object.

10. The method of claim 8 wherein each proxy server contains an array membership list and wherein the act of providing an array membership list at the clieot comprises the acts of:

periodically requesting a oew array membership list from a raodomly choseo proxy server that is listed io a curreot array membership list stored at the client system; and

updating the current array membership list with the new array membership list.

11. A computer-readable medium haviog computer-executable instructions for performiog the steps recited io claim 8.

12. Io a clieot system associated with ao array of proxy servers coofigured so as to provide a cache of uniform resource locator ("URL") data objects distributed across the

array of proxy servers, a method of transmitting a request for a URL data object to a single proxy server of the array that is assigned to store the URL data object, the method comprising the acts of:

providing an array membership list at the client system, the list including information identifying all proxy servers that are active in the array at a given time, the list being located on each proxy server of the array, said act of providing an array membership list comprising the acts of:

periodically requesting the array membership list located at a proxy server of the array; and

updating the array membership list at the client system using the requested array membership list;

in response to a URL request generated at the client system, determining which single proxy server is assigned to store a URL data object associated with the URL request without making a query to any of the proxy servers of the array, by performing the acts of:

for each of tbe proxy servers that are active in the array, computing a first deterministic hash value based on the information identifying the proxy server;

computing a second deterministic hash value of a URL associated with the URL request;

combining, for each of the proxy servers active in the array, tbe first deterministic hasb value and the second deterministic hash value to generate a deterministic combined hash value for each of the proxy servers; and

based on the relative values of the deterministic combined hash values, deterministically identifying which single proxy server is assigned to store the URL data object; and

forwarding the URL request to the identified single proxy server.

13. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 12.

14. A computer-readable medium having computer-executable components for routing a uniform resource locator ("URL") generated at a client system to a single proxy server that is assigned to store a URL data object associated with the URL request, the single proxy server being part an array of proxy servers configured so as to provide a distributed cache of URL data objects, the components comprising:

a request generating component for originating or forwarding a URL request that includes a URL associated with a URL data object that is to be accessed by the client system;

a proxy server identification component for deterministically identifying which single proxy server in the array is assigned to store the URL data object associated witb the URL without sending queries to the all of the proxy servers in the array, including performing the acts of:

for each of the proxy servers that are active in the array, computing a first deterministic hash value of a name of the proxy server;

computing a second deterministic hash value of the URL;

combining, for each of the proxy servers active in the array, the first deterministic hash value and the second deterministic hash value to generate a deterministic combined hash value associated with each of the proxy servers active in the array; and

based on the relative values of the deterministic combined hash values, deterministically identifying which single proxy server is assigned to store the URL data object; and

a routing component to route the generated URL request to the identified single proxy server.

15. A system of server computers configured into a distributed cache for holding Uniform Resource Locator ("URL") data objects and client computers that access the distributed cache, wherein a URL request generated at a client computer can be routed directly to the server computer that is assigned to store the requested URL data object without queries being made to any other server computer, the system comprising:

at least two server computers, each server computer comprising:

a CPU;

storage means, electronically coupled and responsive to said CPU, said storage means containing membership information including information identifying all server computers included in the distributed cache;

means, electronically coupled and responsive to said CPU, for receiving URL requests from clients of the distributed cache; and

at least one client computer being a client of the distributed cache, said client computer comprising:

a CPU;

storage means, electronically coupled and responsive to said CPU, said storage means containing membership information including information identifying all server computers included in the distributed cache;

means, electronically coupled and responsive to said CPU, for generating a URL request;

means, electronically coupled and responsive to said CPU, for deterministically identifying which single server computer in the distributed cache is assigned to store a URL data object associated with the URL request by performing tbe acts of:

for each of the servers computers in the array, computing a first deterministic hash value of the information identifying the server computer;

computing a second deterministic hash value of a URL that is associated with the URL request;

combining, for each of the server computers, the first deterministic hash value and the second deterministic hasb value to generate a deterministic combined hash value associated with each of the server computers; and

based on the relative values of the deterministic combined hash values, deterministically identifying which single server computer is assigned to store the URL data object; and

means, electronically coupled and responsive to said CPU, for routing the URL request to the identified single server computer.

16. A method as recited in claim 1, wherein the act of combining the results of the first deterministic function with the results of the second deterministic function to generate a value for each server comprises the act of using a third deterministic function selected so as to assign the data objects substantially evenly among the servers in the array.

17. A method as recited in claim 1, wherein the act of combining the results of the first deterministic function witb the results of the second deterministic function to generate a value for each server comprises the act of using a third deterministic function selected so as to assign the data objects with relative load factors among the servers in the array.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,311,216 B1
DATED : October 30, 2001
INVENTOR(S) : Brian J. Smith and Hans Hurvig

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 7,
Line 8, change "data base" to -- datebase --
Line 41, after "object access" delete [s]

Column 21,
Line 10, after "that" change "may" to -- many --

Signed and Sealed this

Seventh Day of May, 2002

Attest:

JAMES E. ROGAN
Director of the United States Patent and Trademark Office

Attesting Officer

US006389462B1

(12) **United States Patent**
Cohen et al.

(10) **Patent No.: US 6,389,462 B1**
(45) **Date of Patent: May 14, 2002**

(54) **METHOD AND APPARATUS FOR TRANSPARENTLY DIRECTING REQUESTS FOR WEB OBJECTS TO PROXY CACHES**

(75) Inventors: **Ariel Cohen**, Berkeley Heights; **Sampath Rangarajan**, Bridgewater; **Navjot Singh**, Morristown, all of NJ (US)

(73) Assignee: **Lucent Technologies Inc.**, Murray Hill, NJ (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/212,980**

(22) Filed: **Dec. 16, 1998**

(51) **Int. Cl.**[7] **G06F 15/16**

(52) **U.S. Cl.** 709/218; 709/201; 709/203; 709/219; 709/229; 709/231; 709/239; 707/10; 707/501

(58) **Field of Search** 709/218, 203, 709/219, 201, 224, 229, 239, 231; 707/10, 501; 713/201

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,774,660 A	*	6/1998	Brendel et al.	709/201
5,838,916 A	*	11/1998	Domenikos et al.	709/219
5,864,852 A	*	1/1999	Luotonen	707/10
5,905,872 A	*	5/1999	DeSimone et al.	709/245
5,941,954 A	*	8/1999	Kalajan	709/239

(List continued on next page.)

OTHER PUBLICATIONS

ACEdirector 2 Data Sheet, ACElerate Layer 4 Services, Server Switching Services, http;//www.alteon.com.

"Deploying Transparent Caching With Inktomi's Traffic Server", http://www.inktomi.com.

Danzig, Peter, Network Appliance, Inc., "NetCache Architecture and Deployment", http://www.network application.com.

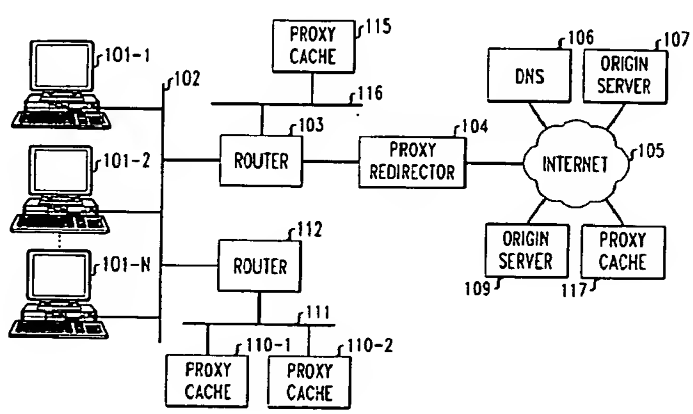
(List continued on next page.)

Primary Examiner—Ayaz Sheikh
Assistant Examiner—Frantz B. Jean
(74) *Attorney, Agent, or Firm*—Stephen M. Gurey

(57) **ABSTRACT**

In order to transparently redirect an HTTP connection request that is directed to an origin server (107) to a proxy cache (110-1), a proxy redirector (104) translates the destination address of packets directed to the origin server to the address of the proxy. During a handshaking procedure, a TCP connection is transparently established between the client (110-1) and the proxy cache. When the client transmits a GET request to what it thinks is the origin server, which request specifies the complete address of an object at that origin server that it wants a copy of, the proxy redirector modifies the complete address specified in that GET request before it is sent to the proxy cache. Specifically, the IP address of the origin server found in the destination field in the IP header of the one or more packets from the client containing the GET request is added by the proxy redirector as a prefix to the complete URL in the GET request to form an absolute URL. The proxy cache determines from that absolute URL whether it has the requested object stored in its cache. If it does, it sends the object back to the proxy redirector, which masquerades those packets as coming from the origin server by translating their destination address to the address of the client and their source address to that of the origin server. If the proxy does not have the requested object, a separate TCP connection is established between the proxy and the origin server from where the object is retrieved and then forwarded over the TCP connection between the client and the proxy. In order to account for the additional number of bytes in the GET request, an acknowledgement sequence number in packets returned from the proxy that logically follow receipt of the GET request are decremented by that number by the proxy redirector before being forwarded to the client. Similarly, a sequence number in packets transmitted by the client subsequent to the GET request are incremented by that number before being forwarded by the proxy redirector to the proxy cache.

55 Claims, 4 Drawing Sheets



U.S. PATENT DOCUMENTS

5,987,523 A	* 11/1999	Hind et al.	....	709/245
6,065,058 A	* 5/2000	Hailpern et al.	....	709/231
6,081,829 A	* 6/2000	Sidana	....	709/203
6,081,900 A	* 6/2000	Subramaniam et al.	.....	713/201
6,098,108 A	* 8/2000	Sridhar et al.	....	709/239
6,112,212 A	* 8/2000	Heitler	....	707/501
6,138,162 A	* 10/2000	Pistriotto et al.	....	709/229
6,189,030 B1	* 2/2001	Kirsch et al.	....	709/224

OTHER PUBLICATIONS

"High-Performance Web Caching White Paper", CacheFlow, Document Center, http://www.cacheflow.com.

"Shared Network Caching and Cisco'Cache Engine", http://www.cisco.com.

"The Content Smart Internet", ArrowPoint, ArrowPoint Communications, 235 Littleton Road, Westford, MA 01886, http://www.arrowpoint.com.

Danzig, P. and Swartz, K. L., "Transparent, Scalable, Fail-Safe Web Caching", Network Appliance, Inc., http://www.networkappliance.com.

RND Networks Inc., "Cache Directing Technology—White Paper", http://www.rndnetworks.com.

Foundry Products, "ServerIron Server Load Balancing and Transparent Caching Swithch", http://www.foundrynet.com.

* cited by examiner

FIG. 1

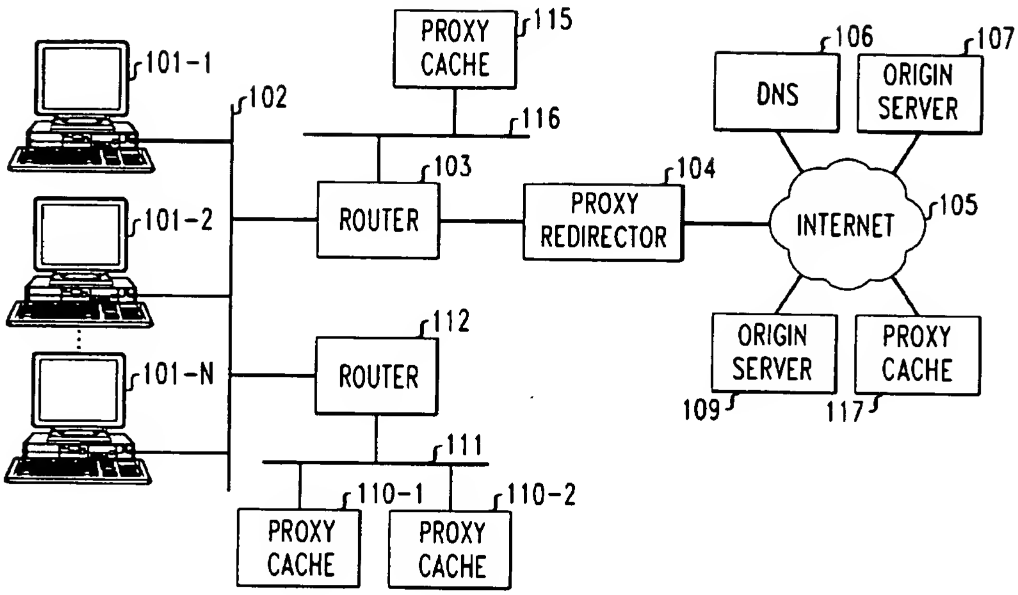

PROXY CACHE
115
106
107
DNS
ORIGIN SERVER
101-1
102
116
103
104
101-2
ROUTER
PROXY REDIRECTOR
INTERNET
105
112
ORIGIN SERVER
PROXY CACHE
101-N
ROUTER
109
117
111
110-1
110-2
PROXY CACHE
PROXY CACHE

FIG. 3

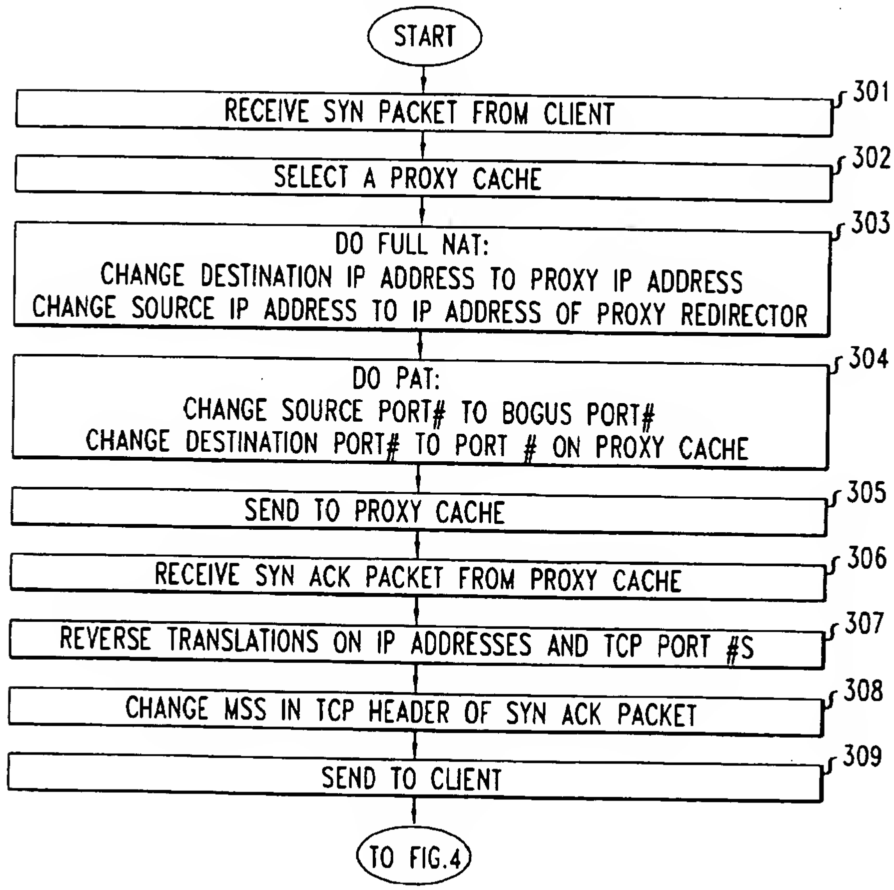

START
RECEIVE SYN PACKET FROM CLIENT
301
SELECT A PROXY CACHE
302
DO FULL NAT:
CHANGE DESTINATION IP ADDRESS TO PROXY IP ADDRESS
CHANGE SOURCE IP ADDRESS TO IP ADDRESS OF PROXY REDIRECTOR
303
DO PAT:
CHANGE SOURCE PORT# TO BOGUS PORT#
CHANGE DESTINATION PORT# TO PORT # ON PROXY CACHE
304
SEND TO PROXY CACHE
305
RECEIVE SYN ACK PACKET FROM PROXY CACHE
306
REVERSE TRANSLATIONS ON IP ADDRESSES AND TCP PORT #S
307
CHANGE MSS IN TCP HEADER OF SYN ACK PACKET
308
SEND TO CLIENT
309
TO FIG.4

FIG. 2

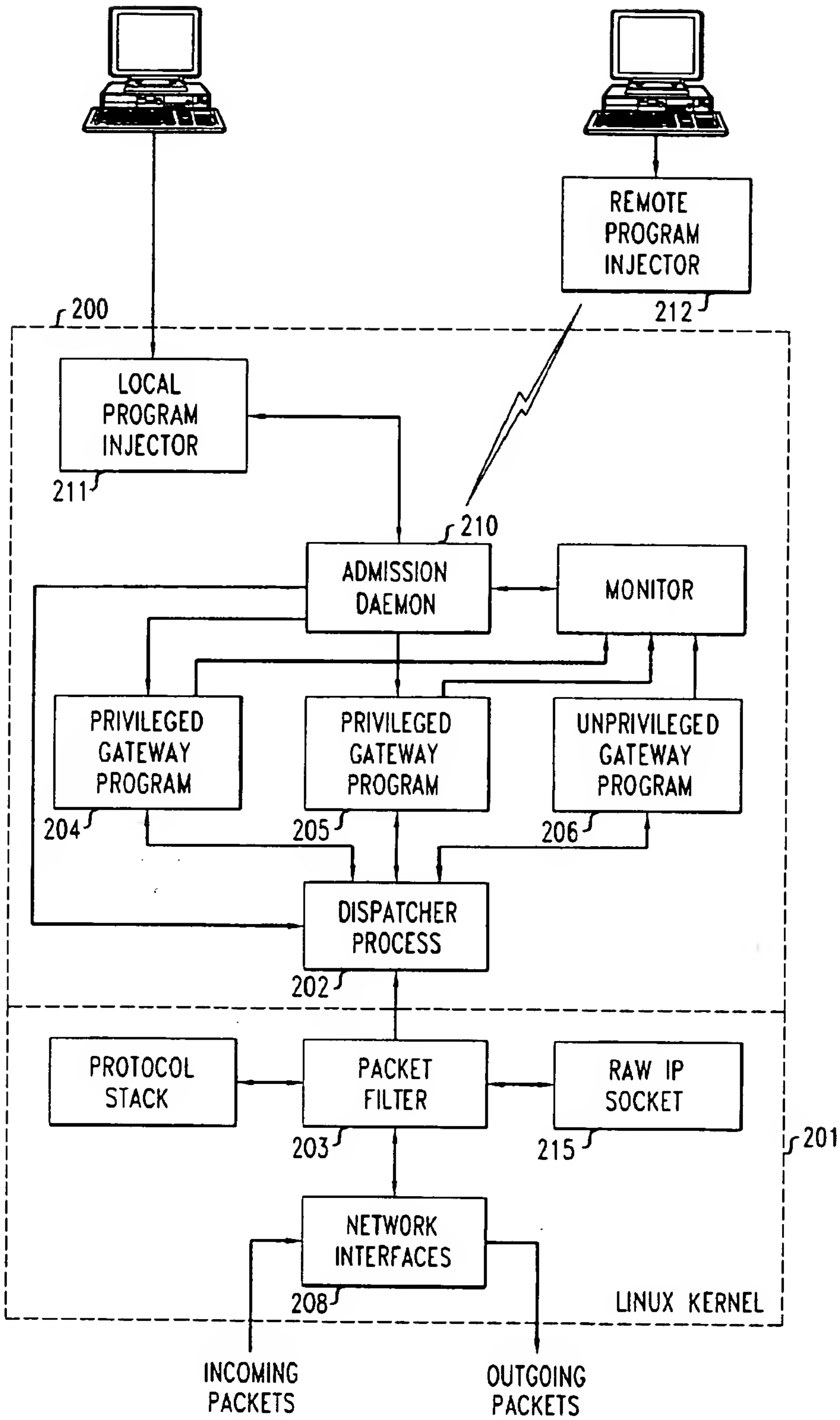

REMOTE PROGRAM INJECTOR
212
200
LOCAL PROGRAM INJECTOR
211
210
ADMISSION DAEMON
MONITOR
PRIVILEGED GATEWAY PROGRAM
204
PRIVILEGED GATEWAY PROGRAM
205
UNPRIVILEGED GATEWAY PROGRAM
206
DISPATCHER PROCESS
202
PROTOCOL STACK
PACKET FILTER
203
RAW IP SOCKET
215
201
NETWORK INTERFACES
208
LINUX KERNEL
INCOMING PACKETS
OUTGOING PACKETS

FIG. 4

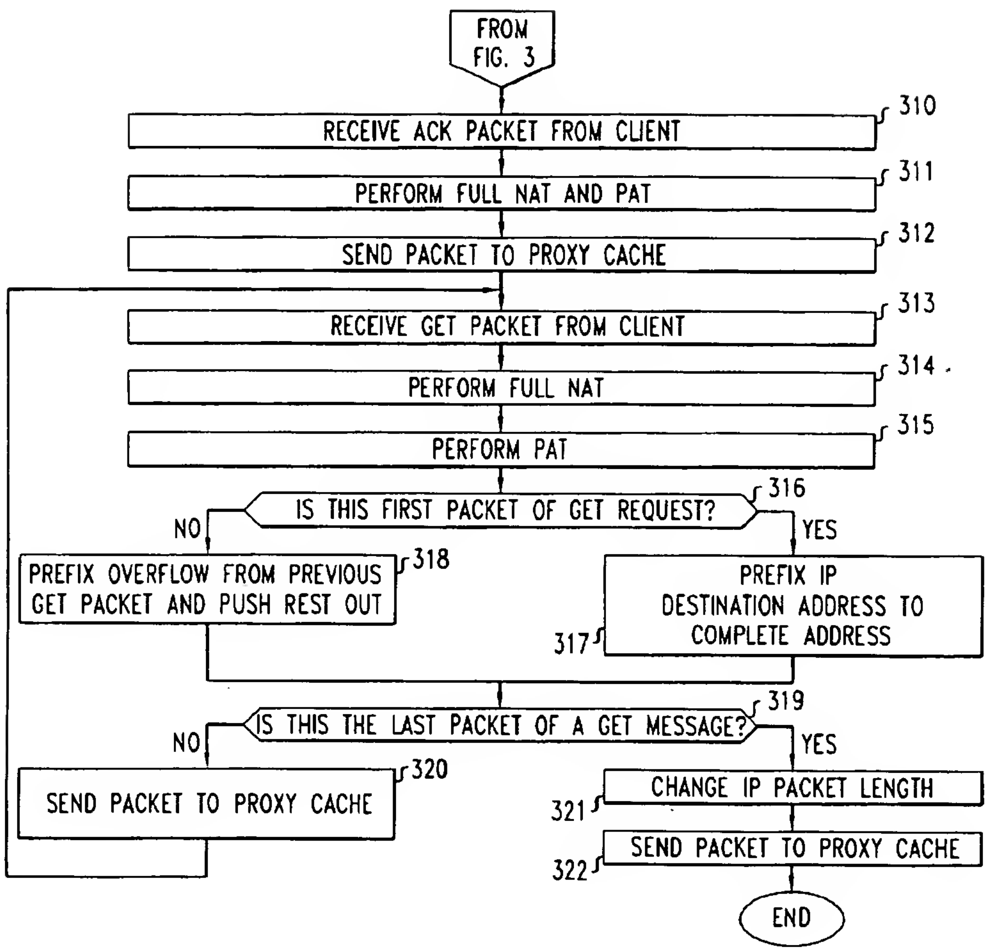

FROM
FIG. 3
RECEIVE ACK PACKET FROM CLIENT
310
PERFORM FULL NAT AND PAT
311
SEND PACKET TO PROXY CACHE
312
RECEIVE GET PACKET FROM CLIENT
313
PERFORM FULL NAT
314
PERFORM PAT
315
IS THIS FIRST PACKET OF GET REQUEST?
316
NO
YES
PREFIX OVERFLOW FROM PREVIOUS GET PACKET AND PUSH REST OUT
318
PREFIX IP DESTINATION ADDRESS TO COMPLETE ADDRESS
317
IS THIS THE LAST PACKET OF A GET MESSAGE?
319
NO
YES
SEND PACKET TO PROXY CACHE
320
CHANGE IP PACKET LENGTH
321
SEND PACKET TO PROXY CACHE
322
END

FIG. 5

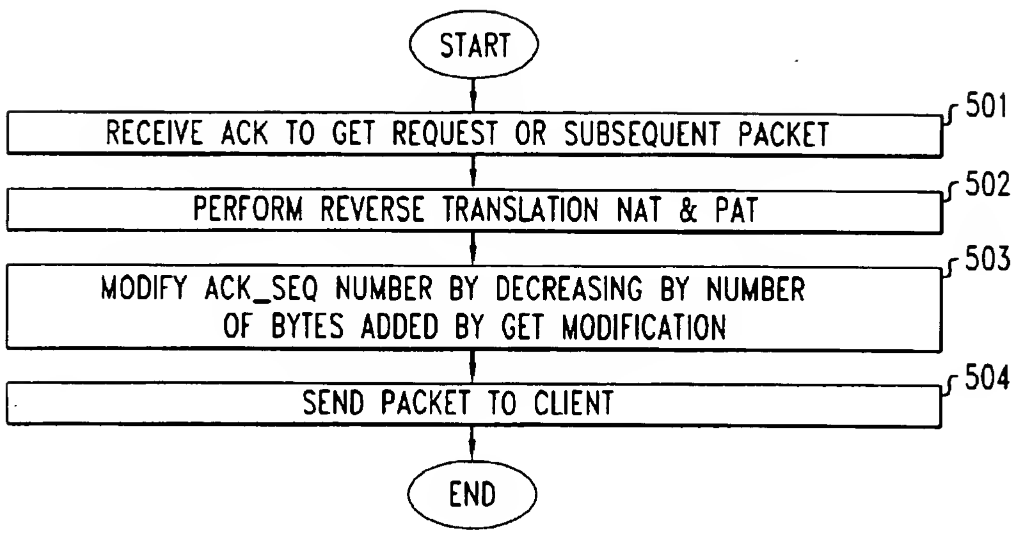
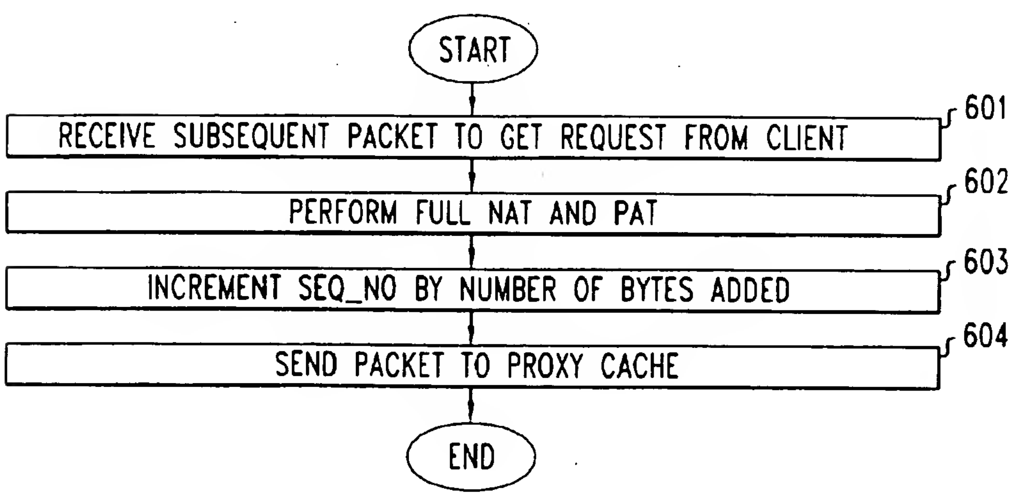

START
RECEIVE ACK TO GET REQUEST OR SUBSEQUENT PACKET
501
PERFORM REVERSE TRANSLATION NAT & PAT
502
MODIFY ACK_SEQ NUMBER BY DECREASING BY NUMBER OF BYTES ADDED BY GET MODIFICATION
503
SEND PACKET TO CLIENT
504
END

FIG. 6


START
RECEIVE SUBSEQUENT PACKET TO GET REQUEST FROM CLIENT
601
PERFORM FULL NAT AND PAT
602
INCREMENT SEQ_NO BY NUMBER OF BYTES ADDED
603
SEND PACKET TO PROXY CACHE
604
END

METHOD AND APPARATUS FOR TRANSPARENTLY DIRECTING REQUESTS FOR WEB OBJECTS TO PROXY CACHES

FIELD OF THE INVENTION

This invention relates to packet-switched computer networks, and more particularly, to a method and apparatus in such a network for transparently intercepting client web requests and redirecting them to proxy caches.

BACKGROUND OF THE INVENTION

Proxy caching is currently used to decrease both the latency of object retrieval and traffic on the Internet backbone. As is well known, if a proxy cache has stored a copy of an object from an origin server that has been requested by a client, the requested object is supplied to the client from the proxy cache rather than from the origin server. This, therefore, obviates the need to send the request over a wide area network, such as the Internet, to the origin server where the original object is stored and the responsive transmission of a copy of the requested object back over the network to the requesting client.

Direction of a request from a client to a proxy cache to determine whether a requested copy of an object is stored in the cache can be accomplished either transparently or non-transparently to the client. Non-transparent redirection is accomplished through the client's browser program which is configured to send all object requests to a designated proxy cache at a specified address. Generally, a browser can be configured to send all of its client requests to a designated proxy cache if the client is connected on a Local Area Network (LAN), or on an Intranet behind a firewall, where a proxy cache associated with that LAN or Intranet is located. When clients are served by a large Internet Service Provider (ISP), however, it is not advantageous from the ISP's standpoint to allow its subscribers to set their browsers to a specific proxy cache associated with the ISP. A large ISP likely will have many proxy caches in several locations and will thus want to maintain control over which of its several particular proxy caches a client request is directed. Further, if a proxy cache whose address is statically set in a client's browser becomes inoperative, all client requests will fail.

It is therefore more desirable from an ISP's standpoint with respect to latency and minimizing traffic onto and off of the network to transparently intercept a client's web request and send it to one of its operative proxy caches to determine whether a copy of the requested object is stored there. If a copy of the requested object is then found to be stored in that proxy cache, a copy of the object is sent to the client, which is unaware that it has been served an object from the proxy cache rather than from the origin server to which it made the request. If the proxy cache does not hold a copy of the requested object, then a separate connection is established between the proxy cache and the origin server to obtain a copy of the object, which when returned to the proxy is sent to the client over the connection established between the client and the proxy.

When a client specifies a URL of the object it is requesting a copy of, a Domain Name Server (DNS) look-up is performed to determine from the URL an IP address of an origin server which has that requested object. As a result of that look-up, an IP address is returned to the client of one of what may be several substantially equivalent servers that contain that object. The client then establishes a TCP connection to that server using a three-way handshake mechanism. Such a connection is determined at each end by a port number and an IP address. First, a SYN packet is sent from the client to that origin server, wherein the destination IP address specified in the packet is the DNS-determined IP address of the origin server and the destination port number for an HTTP request is conventionally port **80**. The source IP address and port number of the packet are the IP address and port number associated with the client. The client IP address is generally assigned to the client by an ISP and the client port number is dynamically assigned by the protocol stack in the client. The origin server then responds back to the client with an ACK SYN packet in which the destination IP address and destination port are the client's IP address and port number and the packet's source IP address and port number are the server's IP address and the server's port number, the latter generally being port **80**. After receipt of the ACK SYN packet, the client sends one or more packets to the origin server, which packets include a GET request. The GET request includes a complete URL, which identifies to that server the specific object within the origin server site that the client wants a copy of. Unlike an absolute URL, which includes both site information (e.g., www.yahoo.com), and object information (e.g., index.html), a complete URL only identifies the particular object (e.g., index.html) that is requested since the packet(s) containing the GET request is sent to the proper origin server site by means of the destination address of the packet(s).

When a browser is configured to non-transparently send all requests to a proxy, a GET request is formulated by the browser that includes the absolute URL of the requested object. That absolute URL is then used by the proxy to establish a separate TCP connection to the origin server if the proxy does not have a copy of the requested object in its cache. The proxy requires the absolute URL since the destination address of the packets to the proxy is set by the browser to the IP address of the proxy rather than the IP address of the origin server. Thus, in order to determine whether it has the object in its cache and if not establish a connection to the origin server, the proxy requires the absolute URL of the origin server in the GET request.

When requests are transparently directed to a proxy cache, however, the client browser is unaware that the request is being directed to the proxy and is possibly being fulfilled from the cache. Rather, the client's browser needs to "think" that it is connected to the origin server to which its SYN and the packet(s) containing the GET request are addressed. Such origin server IP address is determined by the browser through a. DNS look-up. Further, the source address of the ACK SYN packet and the packets containing the requested object must be that same origin server IP address or they will not be recognized by the browser as being the responsive packets to the SYN packet and the request for the object. Thus, in order to transparently send object requests to a proxy cache, a mechanism must be in place along the packet transmission path to intercept an initial SYN packet sent by a browser and to redirect it to the proxy cache to establish a TCP connection. The proxy cache must then masquerade as the origin server when sending the ACK SYN packet back to the client by using the origin server's IP address and port number as the source address of that packet. Further, the subsequent packet(s) containing a GET request must be redirected to the proxy cache and the request fulfilled either from the cache or via a separate TCP connection from the proxy to the origin server. In either case, the source address of packets sent back to the client must be the origin server's IP address and port number to which the packets sent by the client are addressed.

In order for packets associated with a request for an object to be redirected to a proxy cache connected somewhere in

the network, a Layer 4 (L4) switch on the packet path "looks" at the port number of a destination address of a SYN request packet. Since HTTP connection requests are generally directed to port 80 of an origin server, the L4 switch transparently redirects all packets having a port number of 80 in the destination address. The SYN packet is thus sent to a selected proxy cache. In order for the proxy cache to properly respond to the client, as noted, it must know the absolute URL of the requested object and packets returned to the client must masquerade as coming from the origin server. Unlike the non-transparent caching method previously described in which the browser formulates a GET request with the absolute URL, for transparent caching the absolute URL must be provided in some manner to the proxy cache in order for the proxy to determine whether it in fact has the requested object in its cache, or whether it must establish a separate TCP connection to the origin server to request the object. In the prior art, when one or more caches are directly connected to the L4 switch, the switch chooses one of the caches and transparently forwards the packets to that proxy without modifying the source or destination address of the packets. The proxy, working in a promiscuous TCP mode accepts all incoming packets regardless of their destination address. The proxy, then receiving the SYN packet with the origin server's destination address and the client's source address, can respond to SYN packet with an ACK SYN packet. This ACK SYN packet has the client's address as a destination address and a source address masquerading as the origin server address. This packet is transported through the L4 switch onto the network over the TCP connection back to the client. The subsequent packet(s) with the GET request from the client is redirected by the L4 switch to the directly connected proxy. Since the GET packet(s) only contains the complete URL, the proxy must formulate the absolute URL to determine whether its has the requested object in its cache or whether is must establish a separate TCP connection to the origin server. The proxy forms the absolute URL by prefixing the complete URL in the GET request with the IP address of the origin server in the destination address of the packet. The proxy can then determine whether it has the object and, if not, establish a TCP connection to that absolute address. If that particular origin server at that IP address should be inoperative, the proxy can alternatively prefix the complete URL in the GET request with the logical name of the site indicated in the HOST field in the packet(s) containing the GET request.

In the prior art, if the proxy cache is not directly connected to the L4 switch, then the L4 switch must perform a network address translation (NAT) and port address translation (PAT) on those packets directed to port 80 of an origin server. Specifically, when the L4 switch receives a SYN packet to initiate a TCP connection from a client to an origin server, it translates the destination address of the packet from the IP address and port number of the origin server to the IP address and port number of a selected proxy cache. Further, the switch translates the source address of the packet from the clients IP address and port number to its own IP address and a port number. When the proxy responds with an ACK SYN packet, it therefore responds to the L4 switch where a NAT translates the destination IP address from the IP address of the L4 switch to the IP address of the client, and translates the source IP address from the IP address of the proxy to IP address of the origin server. A PAT also translates the port number in the destination address from that of the L4 switch to that of the client, and translates the port number in the source address from that of the proxy to that of the origin server (usually 80). When the client sends an ACK packet and then the packet(s) containing the GET request to the origin server, the L4 switch again performs a NAT, translating the destination IP address to the IP address of the proxy. Thus, when the packet(s) containing the GET request is received by the proxy, it does not know the IP address of the origin server as in the directly connected proxy arrangement described above. The proxy must therefore look at the logical name in the HOST field and perform a DNS look-up to determine that site's IP address. The proxy then uses that IP address in combination with the complete URL in the GET request to form an absolute URL from which it determines whether it has the requested object in its cache. If it doesn't, a separate TCP connection is established from the proxy to that absolute URL to retrieve that object, which is returned to the proxy. Whether the object is found in the proxy cache or is retrieved over the separate connection from the origin server, it is forwarded back to the L4 switch where a NAT and PAT are performed to translate the destination address to that of the client and to translate the source address to the particular origin server to which the client's request was directed. It should be noted that the source address of the origin server obtained when the client's browser initiates a DNS look-up using the origin server's absolute URL may not be the same IP address obtained when the proxy performs a DNS look-up using the combination of the site URL in the HOST field and the complete URL in the GET request.

The above described techniques for performing transparent proxy caching have several disadvantages. Firstly, use of a HOST field to specify a logical name of an origin server is not currently incorporated within the presently employed HTTP1.0 standards. Thus, a HOST field may not be present in the packet(s) containing a GET request. Where, as described above, the information in the HOST field is necessary to form an absolute URL to determine whether the proxy cache has the requested object and, if not, to establish a connection to an origin server from the proxy, the absence of the HOST field results in an unfilled request. Secondly, the prior art techniques require the proxy cache to perform the function of forming an absolute URL from the information in the HOST field and in the packet(s) containing the GET request. Thus, standard proxy caches which expect the client's browser to produce the absolute URL cannot be used. A methodology for transparent proxy caching that is transparent to both the client and the proxy is desirable to avoid modification to the program that controls proxy cache operations. Standard proxy caches could thus be employed anywhere in the network without the need for a special implementation.

The above described prior art techniques have even further disadvantages with respect to persistent connections defined by the HTTP1.1 standards. As defined by these standards, a persistent connection enables a client to send plural GET requests over the same TCP connection once that connection has been established between two endpoints. When a prior art transparent proxy cache is interposed on the connection, a client may "think" it has established a persistent connection to the specific origin server determined through the DNS look-up. The connection in reality, however, is transparently diverted by the L4 switch to a proxy cache. The proxy cache, in response to a DNS look-up using the logical name in the HOST field, may be directed to an equivalent origin server at a different IP address. Further, as each subsequent GET request is received by the proxy from the client within the client's perceived persistent connection, each responsive DNS look-up to the logical name may direct a connection to an even different IP address

of an equivalent origin server. As a result, the advantages of a transaction-oriented persistent connection in which a server is capable of maintaining state information throughout the connection, are lost. A methodology is desirable that maintains persistence to the same origin server to which the clients browser is directed, or to a same equivalent origin server throughout the duration of the persistent connection.

SUMMARY OF THE INVENTION

The problems associated with the prior art techniques for transparent proxy caching are eliminated by the present invention. In accordance with the present invention, a switching entity, such as the L4 switch (referred to hereinafter as a proxy redirector), through which the packets flow, is provided with the functionalities at the IP level necessary to transform the complete URL in each GET request transmitted by a client to an appropriate absolute URL. Specifically, the IP address found in the destination field in the IP header of the packet(s) from the client containing the GET request are added as a prefix by the proxy redirector to the complete URL in the GET request. As a result, the complete URL in the GET request is modified to form an absolute URL which, when received by the proxy cache, is directly used to determine if the requested object is stored in the cache and, if not, to establish a separate TCP connection to the origin server. The GET request received by the proxy is thus equivalent to what it would expect to receive if it were operating in the non-transparent mode. Advantageously, if a persistent connection is established, each subsequent GET request has the same IP address prefix determined by the initial DNS look-up by the client.

By modifying the GET request at the proxy redirector to include the destination address of the origin server, the number of bytes at the IP level in the packet containing the resultant absolute address are increased by the number of bytes in the prefix. Included in the header within each packet is a sequence number (seq) that provides an indication of the position of the first byte number in the payload. Thus, when the IP address is added to a packet, the sequence number of each of the subsequent packets needs to be incremented by the count of the added bytes. Further, an acknowledgement sequence number (ack_seq) in the header on the packets returned from the proxy or the origin server that logically follow receipt of the GET packet(s) at the origin server needs to be decremented by the proxy redirector before being forwarded to the client to avoid confusing the client with respect to what the sequence number of the next byte it sends should be. Further, if the GET request sent by the client encompasses more than one TCP segment, then the extra bytes in the first of the segments caused by the additional bytes added to the URL are shifted into the second segment, and the resultant now extra bytes in the second segment are shifted into the third segment, etc., until the last of the segments. In order to preclude the necessity of requiring an extra segment to be added to the GET request to accommodate the extra bytes, the client sending the GET request, is deceived into sending segments whose maximum size is less than what can actually be received by the proxy as indicated by a maximum segment size (MSS) field in packets from the proxy. The proxy redirector, upon receipt of the On, ACK SYN packet from the proxy, reduces the MSS parameter received from the proxy by the amount of the number of bytes that will be added to the GET request before that parameter is forwarded to the client. Thus, when the client next sends a GET request, each segment is limited to the reduced MSS, thereby insuring that the segment size of a last segment in a GET request after the IP address is prefixed by the proxy redirector to form the absolute URL (whether the GET request is one or more segments long) is less than or equal to the actual MSS that the proxy can receive.

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 is a block diagram of a network that includes a proxy redirector that transparently sends requests from a client to a proxy cache by changing the destination address of packets in a client request from that of the origin server to that of a proxy and the source address from that of the client to that of the switching entity and, in accordance with the present invention, modifies a GET request to include the destination address of the origin server;

FIG. 2 is a block diagram showing the proxy redirector implemented on a programmable network element that manipulates packets in accordance with instructions provided by a loaded program; and

FIGS. 3, 4, 5 and 6 are flow charts detailing the operation of the proxy redirector.

DETAILED DESCRIPTION

With reference to FIG. 1, a plurality of clients 101-1–101-N are connected to a local area network (LAN) 102, such as an Ethernet. LAN 102, which, in turn, is connected through a router 103 to a Level 4 (L4) switch 104 (proxy redirector) which interfaces the LAN with a wide area network (WAN) 105, such as the Internet. Although shown as two separate elements, the functionalities of router 103 and proxy redirector 104 can in actual practice be combined in a single unit. All requests from any of the clients connected to LAN 102 for objects stored in servers connected to the Internet 105 traverse proxy redirector 104 onto the Internet. The packets comprising such requests, which include the standardized packets that establish a TCP connection, are directed to an IP destination address and port number indicated in the IP header of each packet originating from a client source address that includes a client IP address and port number. Similarly, responses to such requests from an origin server connected to Internet 105 are directed via an IP destination address that is equal to the client's IP address and port number from which the request originated, and have as a source address the server's IP address and port number. All such packets directed to any of the clients 101-1–101-N from any server connected to Internet 105 pass through proxy redirector 104.

When any of the clients connected to LAN 102, such as client 101-1, makes a request through a browser for an object by specifying a logical URL, a domain name server (DNS) 106 connected locally or on Internet 105, as shown, is accessed to perform a database look-up based on that logical name. An associated IP address is then returned to the browser. The IP address returned to the browser is the IP address of a particular origin server which contains the 5 object requested through the logical URL. Since a logical name may in fact be associated with a plurality of essentially equivalent origin servers, such as servers 107 and 109, the particular IP address returned to the client browser chosen by DNS 106 may be determined in a round-robin manner. When DNS 106 selects an origin server corresponding to the logical URL, the IP address of the selected origin server, such as, for example, the IP address of origin server 107, is returned to the browser in the requesting client 101-1. That IP address then serves as the IP address to which packets directed to the origin server from the client are directed. Conventionally, http requests are usually directed to port 80 of an origin server.

With the IP address nf the origin server determined and returned to the client, the browser establishes a TCP connection between the client and the origin server through a three-way handshaking process. Specifically, a SYN packet, addressed to the IP address of the selected origin server, is sent by the client. Handshaking is completed when the client receives an acknowledgement of receipt of that SYN packet through an ACK SYN packet sent by that origin server, and responds with a ACK packet to the nrigin server. The browser then sends a GET request that specifies the particular requested object.

In accordance with the present invention, once the IP address of the nrigin server correspnnding to the logical URL name is determined through the DNS look-up, proxy redirector **104**, rather than establishing a TCP connection to the origin server at the determined IP address, transparently establishes a TCP connection between the client and a proxy. If the requested object is stored in the cache, a copy of that object is transparently returned to the requesting client. A TCP connection, therefore, is not established over the Internet **105**, to the actual origin server **107** to provide the requested object to the requesting client. The coss of transmitting the request to the origin server over the Internet and transmitting the copy of the requested object back over the Internet are thereby saved in addition to the time required for transmitting such a request over the Internet and waiting for a response from the origin server. If the proxy cache to which the request is directed does not contain the requested object, a separate TCP connection is established between the proxy cache and the origin server to obtain a copy of the requested object. When the proxy cache then receives the copy of the requested object from an origin server over that separate TCP connection, the copy is forwarded to the client over the original TCP connection that was established between the client and the proxy cache.

In the embodiment shown in FIG. 1, a proxy cache **110-1** is illustratively shown connected to a LAN **111**, which is connected to LAN **102** through a router **112**. Another proxy cache **115** is shown connected on a different LAN **116** through router **103**. Other proxy caches can be located anywhere on LANs **102, 111**, or **116**, on another LAN connected to the Internet **105** such as proxy cache **117**. Proxy redirector **104** selects one of the available proxy caches to which to forward client requests based on a metric such as least-loaded or round-robin, based on IP header information such as the origin server IP address. With respect to the latter, all objects from a specific origin server will be served by a specific proxy.

In the preferred embodiment described herein, proxy redirector **104** includes a programmable network element of the type described in co-pending U.S. patent application Ser. No. 09/190,355, filed Nov. 12, 1998, which application is incorporated herein by reference. As described in that application, that programmable network element in the preferred embodiment runs on a Linux machine. As shown in FIG. 2, the programmable network element **200** includes a dispatcher process **202** with which plural different gateway programs (**204, 205** and **206**) register and request access to IP packets that fit specific descriptions. Such programs are loaded through an admission daemon **210** via a local program injector **211** or a remote program injector **212**. A gateway program, for example, can request access to incoming packets to network interface **208** that match certain source and destination IP address ranges and port numbers. The dispatcher process **202** uses a packet filter **203** in the Linux kernel **201** to obtain packets requested by the gateway programs and uses a raw IP socket **215** to send packets that have been manipulated in accordance with the gateway program back to the kernel for output back to the network through filter **203** thrnugh network interfaces **208**. Library functions are provided in the programmable network element that enable a gateway program to communicate with the dispatcher process **202** to register rules that specify the type of IP packets that a gateway program wants diverted to it. A gateway program can request either a complete IP packet or only the IP and TCP header of a packet and can change both the header and payload of a packet.

In the present invention, that programmable network element is nperative in combination with a gateway program that manipulates the destination and source addresses of packets flowing there through in a manner to be described, as well as modifying, as will be described, information in the packet(s) containing the GET request that specifies the URL of the requested object. Specifically, the programmable network element in combination with the gateway program operates on packets associated with HTTP requests, which are determined from the destination port number. As previously noted, HTTP requests are conventionally addressed to port **80** of an origin server. Thus, the programmable network element/gateway program which together comprise proxy redirector **104** in this embodiment, captures through the dispatcher process of the programmable network element, packets directed to port **80** and then performs address translations on those captured packets to readdress these packets to a selected proxy. With respect to address translations, the gateway program translates the destination IP address of packets addressed to the origin server to the IP address of a selected proxy cache and translates the source IP address of such packets from that of the client to the IP address of proxy redirector **104**. Further, in order for proxy redirector **104** to identify requests from plural client terminals that are directed to the same proxy, the source port number is translated to a bogus ghost port number at the proxy redirector. Thus, when proxy cache responds, the packets transmitted by the cache have a destination IP address of proxy redirector **104** at that bogus port number, which is distinctly associated with the client. The gateway program within proxy redirector **104** then translates the IP destination address of these responsive packets from the proxy to the IP address of the client and translates the bogus destination port number to the port number from which the client originated its request. Further, the gateway program translates the source IP address of such responsive packets from that of the proxy to the IP address of the origin server and the port number to the port (**80**) to which the client's requests were originally directed. Thus, the packets which are returned to the client from the proxy masquerade as if they had originated from the origin server to which the client "believed" its request had been sent.

By performing the above-described network address translations (NATs) and port address translations (PATs), packets from a client **101-1** are transparently directed by proxy redirector **104** to a proxy cache. Responsive packets from the proxy cache are sent to proxy redirector **104** where they are redirected to client **101-1**.

In establishing a TCP connection that is directed to an origin server, client **101-1** first transmits a SYN packet, which is intercepted by proxy redirector **104**. Proxy redirector **104** selects a proxy cache, such as proxy **110-1**, to redirect this request and creates a connection control block (CCB) to maintain information about the connection. Selection of the particular proxy is determined, as described above, by one of several possible algorithms. The CCB is used to store the client IP address and TCP port number and

the origin server IP address and TCP port number, both of which are contained in the IP header of the SYN packet, and the chosen proxy's IP address. The destination address is then changed to that of the chosen proxy and the packet is sent back to the network for redirection to its new destination address of the proxy **110-1**. All subsequent packets that originate from the same client with the same TCP port number are then forwarded to the same proxy. Proxy **110-1** responds with an ACK SYN packet which is directed via its destination address to proxy redirector **110-1**. Proxy redirector **104** then translates the source IP address and port number to those of the origin server and the destination IP address and port number to those of the client. When the packet arrives at the client the client believes that it is connected to the origin server. The client then responds with an ACK packet to the origin server, which is redirected by proxy redirector **104** to proxy cache **110-1**, to complete the handshaking process.

After the TCP connection is established between client **101-1** and proxy cache **110-1**, client **101-1** sends one or more packets containing a GET request addressed to the origin server. Such packets are thus "captured" by proxy redirector **104** and redirected to proxy cache **110-1**. As previously discussed, the GET request sent by the client contains only the complete URL sent by the client browser which in itself provides insufficient information for the proxy cache to determine whether it has the requested object and, if not, to forward it to the origin serve,r which does. In accordance with the present invention, the gateway program that is operative with the programmable network element of the proxy redirector **104**, captures this packet or packets and, in addition to previously described address translations, transforms the complete URL to an absolute URL by prefixing it with the IP address of the origin server obtained from the destination IP address of the packet(s) containing the GET request. Thus, Level **7** (application) information is modified to assist in level **4** routing.

In order to make the URL transformation transparent to both the client and proxy cache endpoints, changes in IP and TCP headers are also required. Since the GET request modification increases the length of the IP packet that carries the GET request, the total length field on the IP header of this packet is increased by an offset. The offset amount is recorded in the CCB. In addition, the TCP header contains sequence numbers (seq) and acknowledgement sequence numbers (ack_seq) that need to be translated. The seq in the TCP header indicates the byte number of the first byte on this packet going from the sender to the receiver over the TCP session and the ack_seq indicates the byte number of the next byte that the sender expects to receive from the receiver. For all packets after the GET packet(s) that go from the client to the proxy cache, the seq is increased by an offset equal to the lengthof(absolute URL)-lengthof(complete URL) so that the seq matches the byte number of the byte that the proxy cache expects to receive from the client. Similarly, on all packets starting with the acknowledgement to the GET packet that go from the proxy cache to the client through the proxy redirector, the ack_seq is decreased by the same offset so that the ack_seq matches the byte number of the byte that the proxy cache would expect the client to send in the next packet following the GET packet. By performing these changes in the header, the client and proxy cache endpoints remain unaware of the modification in the GET packets from the complete URL to the absolute URL.

Table 1 illustrates the URL and other header transformations performed

TABLE 1

Arriving packet:

```
--->    beginning of packet header dump <---
--->    IP header: version=4 hdr_len=5 TOS=0 pkt_len=346 id=60276
        frag_off=4000H TTL=64 protocol=6 cksum=1792H
        saddr=135.104.25.243 daddr=204.71.200.244
--->    TCP header: sport=1273 dport=80 seq=2189084427 ack_seq=3266449517
        tcp_hdr_len=5 flags=ACK PSH
        res1=0H res2=0H window=31856 cksum=162H urgent=0
--->    beginning of packet data dump <---
GET /a/ya/yahoomail/promo1.gif HTTP/1.0
Referer: http://www.yahoo.com/
Connection: Keep-Alive
User-Agent: Mozilla/4.05 [en] (X11; U; Linux 2.1.103 i686)
Host: us.yimg.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg image/png
Accept-Language: en
Accept-Charset: iso-8859-1, *,utf 8
```

Modified to:

```
--->    beginning of packet header dump <---
--->    IP header: version=4 hdr_len=5 TOS=0 pkt_len=367 id=60276
        frag_off=4000H TTL=64 protocol=6 cksum=1792H
        saddr=135.104.25.245 daddr=135.104.25.31
--->    TCP header: sport=5000 dport=7000 seq=2189084427 ack_seq=3266449517
        tcp_hdr_len=5 flags=ACK PSH
        res1=0H res2=0H window=31856 cksum=162H urgent=0
--->    beginning of packet data dump <---
GET http://204.71.200.244/a/ya/yahoomail/promo1.gif HTTP/1.0
Referer: http://www.yahoo.com/
Connection: Keep-Alive
User-Agent: Mozilla/4.05 [en] (X11; U; Linux 2.1.103 i686)
Host: us.yimg.com
```

TABLE 1-continued

```
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg image/png
Accept-Language: en
Accept-Charset: iso-8859-1, *,utf 8
```

on a GET packet that arrives at proxy redirector **104** from a client **101-1**. The packet is destined to an origin server at a destination IP address (daddr) 204.71.200.244 (www.yahoo.com) at a destination port (dport) **80**, requesting object /a/ya/yahoomail/promo1.gif. As can be noted in the modified packet, the packet is redirected to a proxy cache **110-1** at IP address 135.104.25.31 port 7000 by changing the daddr and dport header information. Also, the complete URL of the object in the GET request is modified in the translated packet by prefixing it with http://204.71.200.244 to form the absolute URL, where that prefix is obtained from the daddr header in the arriving packet. This transformation increases the length of the packet by 21 bytes so that the pkt_len field in the header is modified from 346 to 367 bytes. Further, the source port is modified to a bogus port number by changing sport to 5000.

Table 2 shows the translations performed by the proxy redirector **104** to an acknowledgment from proxy cache **110-1** to the GET request. The arriving

TABLE 2

```
Arriving packet:

--->    beginning of packet header dump <---
--->    IP header: version=4 hdr_len=5 TOS=0 pkt_len=40 id=14559
        frag_off=4000H TTL=255 protocol=6 cksum=10eH
        saddr=135.104.25.31 daddr=135.104.25.245
--->    TCP header: sport=7000 dport=5000 seq=3266449517 ack_seq=2189084754
        tcp_hdr_len=5 flags=ACK
        res1=0H res2=0H window=8433 cksum=32H urgent=0
Modified to:

--->    beginning of packet header dump <---
--->    IP header: version=4 hdr_len=5 TOS=0 pkt_len=40 id=14559
        frag_off=4000H TTL=255 protocol=6 cksum=10eH
        saddr=204.71.200.244 daddr=135.104.25.243
--->    TCP header: sport=80 dport=1273 seq=3266449517 ack_seq=2189084733
        tcp_hdr_len=5 flags=ACK
        res1=0H res2=0H window=8433 cksum=32H urgent=0
```

packet is addressed (daddr) to the IP address of the proxy redirector at tbe bogus port (dport) 5000 used by the proxy redirector for this TCP connection. The source IP address (saddr) and the source port (sport) are those of the proxy cache to wbere the GET request was directed. The ack_seq field indicates tbe byte number of the first byte that is expected to be sent in the packet following the GET packet. In this example, ack_seq is equal to tbe sequence number (seq) 218908427 of the GET packet plus the length of the GET packet, which in this case per Table 1 is 367. Thus ack_seq of the arriving packet is 218908754. Since client **101-1** is unaware tbat a proxy redirector bas increased tbe length of the GET packet by 21 bytes, proxy redirector **104** decreases the ack_seq field by the number of bytes added, 21. Further, proxy redirector **104** translates the destination IP address and port number to those of the client **101-1**, and the source IP address and port number to that of the origin server. The modified packet, when received by the client, thus appears to the client to have originated from the origin server and the ack_seq field indicates a byte number that the client would next expect to send having previously sent a packet of length 367 bytes. All packets that are subsequently sent through the proxy redirector **104** to client **101-1** from proxy cache **110-1** are similarly modified to decrement the ack_seq field by the number of bytes, 21, added to the GET packet.

Table 3 illustrates a next packet sent by client **101-1** to the origin server after the GET packet. In this packet the sequence number (seq) is equal to the modified ack_seq sent to the client, as per Table 2. The destination IP address and port number are those of the origin server and the source IP address and port number are those of the client. Wben received by proxy redirector **104**, tbe packet is modified to cbange the source IP address and port number to tbe IP address and bogus port number of the proxy redirector. The destination address IP and port number are translated to that of proxy cache **110-1**. The sequence number (seq) is increased by that same value of 21 to match the byte number that the proxy cacbe expects to receive based on tbe sequence number previously received in the GET packet and the length, **367**, of the GET packet

TABLE 3

```
Arriving packet:

--->    beginning of packet header dump <---
--->    IP header: version=4 hdr_len=5 TOS=0 pkt_len=40 id=60281
        frag_off=4000H TTL=64 protocol=6 cksum=18bfH
        saddr=135.104.25.243 daddr=204.71.200.244
```

TABLE 3-continued

--->	TCP header: spnrt=1273 dport=80 seq=2189084733 ack_seq=3266450977 tcp_hdr_len=5 flags=ACK res1=0H res2=0H window=31856 cksum=d3f7H urgent=0
Modified to:	
--->	beginning of packet header dump <---
--->	IP header: version=4 hdr_len=5 TOS=0 pkt_len=40 id=60281 frag_off=4000H TTL=64 protocol=6 cksum=18bfH saddr=135.104.25.245 daddr=135.104.25.31
--->	TCP header: sport=5000 dport=7000 seq=2189084754 ack_seq=3266450977 tcp_hdr_len=5 flags=ACK res1=0H res2=0H windnw=31856 cksum=d3f7H urgent=0

received. All subsequent packets directed to the origin server from client **101-1** are similarly modified before being directed to proxy cache **110-1**.

In the above-description, it has been assumed that the length of the GET request both before modification, and after the URL extension is less than the maximum TCP segment size. In fact, the length of the GET request may be longer than one TCP segment. If the length of the GET request carrying the complete URL occupies x number of TCP segment and, after it is modified to carry the absolute URL, it still also fits into that same x number of TCP segments, then the segment carrying the URL is modified and overflowing characters are pipelined from one segment to the next. Thus, the overflowing characters from a previous packet are prefixed to the start of the next packet, etc., until the last packet, which length is increased by the increased number of bytes due to the URL modification. Therefore, the packet length of only the last segment is modified to include the characters that have been shifted into that segment. The ack_seq parameter in packets from the proxy cache to the client is modified starting from the acknowledgment to the last GET packet.

If the modification of the URL to the absolute URL could cause the last TCP segment of the GET request to overflow to another segment, a new TCP segment would need to be constructed and injected by the proxy redirector. The proxy redirector would then need to have the capability to retransmit this segment if it was lost. Thus, the proxy redirector would need to have some TCP layer functionalities. In order to avoid adding higher level functionality to the proxy redirector, segment sizes are limited to less than what the proxy cache is actually capable of receiving. When the complete URL is transformed to an extended URL, the maximum increase in size is 22 bytes, equal to the maximum length of an IP address of 15 bytes plus 7 bytes from the prefix: http://. The client is deceived to send segments whose maximum size is 22 bytes less than what the protocol allows it to send. The TCP segment size sent by the client is determined by what the proxy cache, in its handshake with the client, indicates as the maximum segment size it can receive. This is indicates by the proxy cache through the maximum segment size (MSS) field in the ACK SYN packet. Accordingly, the proxy redirector **104** intercepts the ACK SYN packets and decreases the specified MSS amount by 22. For example, if the MSS specified by the proxy cache is 1460, it is modified to 1438 by the proxy redirector before being sent to the client. When the client next sends a GET request, the TCP segments are limited to 1438 bytes. In the worst case, when the client sends a GET request, 22 bytes will be added to the xth TCP segment that carries this request. The length of this xth TCP segment will still then be within the maximum length specified by the proxy cache. If the event that the proxy cache does not stipulate a maximum MSS in the ACK SYN packet, the default used by the client is 536 bytes. An MSS option is then added by the proxy redirector to inform the client that the maximum MSS expected by the other end of the TCP connection is 514 bytes.

As previously described, a NAT and PAT are performed by proxy redirector **104** on all packets addressed by client **101-1** to an origin server, and all packets addressed by proxy cache **110-1** to proxy redirector **104** for return to the client. Proxy redirector **104** thus performs a NAT and a PAT on these packets flowing in both direction. If proxy redirector **104** selects a proxy cache that is located in such a point on the network that packets from the proxy cache addressed directly to client **101-1** must pass through proxy redirector **104** due to the network configuration, then proxy redirector need only perform a half-NAT on the packets flowing through it. Specifically, if proxy redirector **104** selects a proxy cache such as proxy cache **117**, all packets addressed to client **101-1** must pass through proxy redirector **104**. Proxy redirector **104** thus only needs to transform the destination IP address and port number of packets from client **101-1** to the IP address and port number of proxy cache **117**, while maintaining the source IP address and port number as those of client **101-1**. The packets returned from proxy cache **117** will thus be addressed to the client's IP address and port number. When they pass through proxy redirector **104**, they are captured and the transformation of the source IP address and port to those of the origin server are the only address changes that need to be effected.

The problems of the prior art with respect to persistent connections is obviated in accordance with the present invention. As previously noted, during a persistent connection plural GET requests can be made by a client. In the prior art, as described, each GET request can result in a connection from a proxy cache to a different origin server if the proxy cache does not have the requested objects. The ability of a server to maintain the state of a client's connection throughout the duration of the connection is compromised if each GET request results in connections to multiple servers. In accordance with the present invention, once the IP address of the origin server is determined at the initial DNS lookup, that same IP address is used by the proxy redirector as a prefix to each complete URL in every GET request issued by the client throughout the duration of the persistent connection. Thus, assuming the proxy cache does not contain any of the requested objects, the proxy cache will establish a TCP connection to the same origin server in response to each GET request generated by the client. It should be noted that if plural client GET requests are forwarded by the proxy redirector to a proxy cache within a persistent TCP connection, ack_seq parameter in packets that flow through

the proxy redirector from the proxy following each GET request must reflect the cumulative changes effected by translating the complete URL to the absolute URL in each of the preceding GET requests within the same TCP connection. Similarly, in all packets received by the proxy redirector from the client directed to the origin server within a persistent TCP connection, the seq parameter must reflect cumulative changes.

FIGS. 3, and 4, together are flow charts detailing the functions of proxy redirector **104** in establishing a TCP connection to a proxy cache and modifying the GET request so that such requests can be transparently forwarded to the proxy. At step **301**, a SYN packet arrives from the client at the proxy redirector At step **302**, proxy redirector selects a proxy cache based on a load balancing algorithm or on an arbitrary or random selection. At step **303**, proxy redirector performs a full NAT, changing the daddr from that of the origin server to that of the selected proxy and saddr from that of the client to that of the proxy redirector. At step **304** a PAT is performed, changing sport to that of a bogus ghost port number and dport to the proxy's port number. At step **305**, the SYN packet is sent to the proxy. In response to that SYN packet, the proxy responds, at step **306**, with a SYN ACK packet containing an MSS parameter in the TCP header. At step **307**, a reverse translation is performed on both the IP addresses and port numbers, changing saddr and sport to those of the origin server and daddr and dport to those of the client. At step **308**, the MSS field is changed by reducing the value of the MSS received from the proxy by **22**. At step **309**, the ACK SYN packet is sent to the client. At step **310**, proxy redirector receives a responsive ACK packet from the client. At step **311**, a full NAT and PAT are performed on that packet and, at step **312**, the modified packet is sent to the proxy, thereby completing the handshake sequence.

At step **313**, a packet containing a GET request is received from the client. A full NAT is performed at step **314** and a PAT is performed at step **315**. A determination is made at decision step **316** whether this is a first packet in the GET request. If yes, at step **317**, the IP address of the origin server obtained Is from daddr of the arriving packet is prefixed to the complete URL in the GET request. If, at step **316**, the packet is not a first packet in a GET request, then, at step **318**, the overflow bytes from the previous GET packet are prefixed to those bytes in the current packet and if the total number of bytes in the resultant packet is than the actual MSS sent by the proxy, the overflow bytes greater are buffered for prefixing to the next packet. After alternative steps **316** or **318**, at step **319**, a determination is made whether the current packet is the last packet of a GET request. If not, at step **320**, the current packet is sent to the proxy and the flow returns to step **313** to receive the next packet in the GET request from the client. If at step **319**, the current packet is the last packet in a GET message, then, at step **321**, the pkt_len parameter of that packet is changed to reflect the change in length of the packet. At step **322**, the modified packet is sent to the proxy.

FIG. 5 illustrates the steps performed by the proxy redirector for each packet received from the proxy starting from the ACK to the GET request through the end of the connection. At step **501**, the proxy redirector receives the ACK to the GET request, or any other packet that logically follows the ACK to the GET request. At step **502**, reverse NAT and PATs are performed, translating daddr and dport to those of the client and saddr and sport to those of the origin server. At step **503**, ack_seq is decreased by the amount added in the preceding GET request. At step **504**, the modified packet is sent to the client.

FIG. **6** illustrates the step performed by the proxy redirector for each packet destined for the origin server from the client that follows the GET request. At step **601**, a packet subsequent to the GET request is received from the client. At step **602**, a full NAT and PAT are performed. At step **603**, seq_no is increased by the amount of bytes added by modifying the previous GET request. At step **604**, the packet is sent to the proxy.

In the discussion above of FIGS. **3**, **4**, **5** and **6**, it has been assumed that the proxy cache is located in a position such that packets directed to the client will not automatically flow through the proxy redirector. Thus, all packets from the proxy are addressed to the proxy redirector. Therefore, for packets flowing from the client, and packets flowing from the proxy, the proxy redirector performs a full NAT and PAT. If however, as previously described, the proxy cache selected by the proxy redirector is located on the network so that all packets from proxy to the client automatically flow through the proxy redirector, then, in the steps shown in FIGS. **3**, **4**, **5** and **6**, only a half NAT needs to be performed.

Although described in conjunction the programmable network element shown in FIG. **2**, the proxy redirector of the present invention could be implemented through other means, using hardware, software, or a combination of both. As an example, a level 4 switch having a fixed program to perform the required packet manipulations required by the present invention could be used.

As described, the proxy cache returns requested objects to the address from which a request originated as indicated by the saddr and sport parameters in the IP header information, which is the address of the proxy redirector **104** when the proxy cache is not connected on the network so that all responses do not automatically pass through the proxy redirector. The interactions between the requesting client and the proxy cache are transparent to both the client and the proxy cache, since the client does not "know" that its request is being redirected to the proxy by the proxy redirector, and the proxy cache, when receiving a GET request with an absolute URL does not know that that absolute URL is not being formulated by the client's browser operating in a non-transparent mode. Advantageously, the proxy cache requires no software modifications and standard proxy caches, connected anywhere on the network can be used in conjunction with the proxy redirector. If, however, the proxy is modified, using a programmable network element as previously described, for example, the requested object retrieved by the proxy from its own cache or received from an origin server, can be sent directly back to the client, thereby obviating the need to send such packets back to the proxy redirector for address translations and redirection to the client. By performing only a half-NAT at the proxy redirector and leaving the client's saddr and sport as the source IP address and port number in the header of the SYN packet, GET request packet(s), and other packets forwarded by proxy redirector **104** from the client, the proxy cache can return packets responsive to the request directly to the client by substituting the origin server's IP address and port number as the source address for its own address. If the proxy redirector performs a full NAT and PAT, then another mechanism must be incorporated to provide the client address to the proxy cache, such as incorporating the client address information as part of an appendix to the absolute address in the modified GET request and stripping the appended client address information at the proxy before determining whether the requested object is stored in the cache or whether a connection to the origin server need be made. Advantageously, by sending the packets from the

proxy cache directly back to the client, the delay encountered by transmitting such packets back to the proxy redirector for address translation and redirection is eliminated. Disadvantageously, the proxy cache must be modified to perform these functions, precluding use of standard available proxy caches.

Although described hereinabove in connection with GET requests, the present invention can equally be applied to redirection of any type of request message in which the token is, for example, GET, POST or HEAD, or any other type of token yet to be implemented and/or standardized.

The foregoing therefore merely illustrates the principles of the invention. It will thus be appreciated that those skilled in the art will be able to devise various arrangements which, although not explicitly described or shown herein, embody the principles of the invention and are included within its spirit and scope. Furthermore, all examples and conditional language recited hereinabove are principally intended expressly to be only for pedagogical purposes to aid the reader in understanding the principles of the invention and the concepts contributed by the inventors to furthering the art, and are to be construed as being without limitation to such specifically recited examples and conditions. Moreover, all statements hereinabove reciting principles, aspects, and embodiments of the invention, as well as specific examples thereof, are intended to encompass both structural and functional equivalents thereof. Additionally, it is intended that such equivalents include both currently known equivalents as well as equivalents developed in the future, i.e., any elements developed that perform the same function, regardless of structure.

Thus, for example, it will be appreciated by those skilled in the art that the block diagrams and flowcharts described hereinabove represent conceptual views of illustrative circuitry and processes embodying the principles of the invention. Similarly, it will be appreciated that any flowcharts, flow diagrams, state transition diagrams, pseudocode, and the like represent various processes which may be substantially represented in computer readable medium and so executed by a computer or processor, whether or not such a computer or processor is explicitly shown.

The functions of the various elements shown in the FIGS., including functional blocks labeled as "processors" may be provided through the use of dedicated hardware as well as hardware capable of executing software in association with appropriate software. When provided by a processor, the functions may be provided by a single dedicated processor, by a single shared processor, or by a plurality of individual processors, some of which may be shared. Moreover, explicit use of the term "processor" or "controller" should not be construed to refer exclusively to hardware capable of executing software, and may implicitly include, without limitation, digital signal processor (DSP) hardware, read-only memory (ROM) for storing software, random access memory (RAM), and non-volatile storage. Other hardware, conventional and/or custom, may also be included. Similarly, any switches shown in the FIGS. are conceptual only. Their function may be carried out through the operation of program logic, through dedicated logic, through the interaction of program control and dedicated logic, or even manually, the particular technique being selectable by the implementor as more specifically understood from the context.

In the claims hereof any element expressed as a means for performing a specified function is intended to encompass any way of performing that function including, for example, a) a combination of circuit elements which performs that function or b) software in any form, including, therefore, firmware, microcode or the like, combined with appropriate circuitry for executing that software to perform the function. The invention as defined by such claims resides in the fact that the functionalities provided by the various recited means are combined and brought together in the manner which the claims call for. Applicant thus regards any means which can provide those functionalities as equivalent to those shown hereinabove.

What is claimed is:

1. A method at a Layer 4 switch for redirecting an HTTP connection request from a client that is directed to an origin server to a proxy cache over a packet-based computer network comprising the steps of:

receiving at least one packet containing a request message within the HTTP connection, the at least one packet having an IP header comprising a destination address of an origin server, the request message including a complete address of a specified object at the origin server;

modifying, at the packet level, the request message by combining the destination address of the origin server with the complete address; and

forwarding the at least one packet containing the modified request message to the proxy cache over the packet-based network;

wherein the proxy cache is a standard proxy cache and, in the forwarding step, the at least one packet containing the modified request is redirected to the standard proxy cache transparently from the standpoints of both the client and the proxy cache, the proxy cacbe being able to determine whether it can satisfy the request for the specified object by using the combined destination address of the origin server and the complete address contained in the modified request message.

2. The method of claim 1 wherein the step of modifying comprises the step of prefixing the complete address in the request message with the destination address of the origin server.

3. The method of claim 2 further comprising the step of:

translating the destination address in an IP header of the at least one packet containing the modified request message to the address of the proxy cache.

4. The method of claim 3 further comprising the step of:

translating a source address in the IP header of the at least one packet containing the modified request message from an address of the client to an address of the Layer 4 switch.

5. The method of claim 3 further comprising the step of:

in a TCP header in packets received from the proxy cache commencing with an acknowledgment to the modified request message, modifying a value in an acknowledged byte sequence number field to reflect the increased number of bytes in the modified request message resulting from the step of prefixing the complete address with the destination address of the origin server.

6. The method of claim 5 wherein the step of modifying the value in the acknowledged byte sequence number field comprises the step of decreasing the value in the acknowledged byte sequence number field by the number of bytes added by prefixing the destination address of the origin server to the complete address in the request message.

7. The method of claim 6 further comprising the step of:

translating in the IP header the destination address to the address of the client and the source address to that of the origin server in the packets received from the proxy

cache commencing with the acknowledgment to the request message.

8. The method of claim 3 further comprising the step of:

modifying a value in a sent byte sequence number field to reflect the increased number of bytes in the modified request message resulting from prefixing the complete address with the destination address of the origin server in a TCP header in packets received from the client following receipt of the request message.

9. The method of claim 8 wherein the step of modifying the value in the sent byte sequence number field comprises the step of increasing the value in the sent byte sequence number field by the number of bytes added by prefixing the destination address of the origin server to the complete address in the request message.

10. The method of claim 9 further comprising the step of:

translating the destination address to the address of the proxy cache in the IP header in each packet received from the client following receipt of the request message.

11. The method of claim 2 further comprising, prior to the step of receiving the at least one packet containing the request message, the steps of:

receiving a packet from the proxy cache indicating a maximum segment size that packets sent to it thereafter should have;

reducing by a predetermined number the maximum segment size of packets to be thereafter sent to the proxy cache; and

sending a packet to the client indicating the reduced maximum segment size that packets thereafter sent by the client to the origin server should have.

12. The method of claim 11 wherein the predetermined number is equal at least to the maximum number of bytes added by the step of prefixing the destination address to the complete address in the request message.

13. The method of claim 12 further comprising the steps of:

successively shifting to a next packet in the request message overflow bites caused by the step of prefixing the destination address to the complete address in the request message, if the request message is contained within more than one packet; and

changing a length-of-packet parameter in the IP header in a last packet of a multi-packet request message to reflect the bytes shifted into the last packet from a previous packet.

14. The method of claim 12 further comprising the step of changing a length-of-packet parameter in the IP header to reflect the change in the length of that packet caused by the step of prefixing the destination address to the complete address in the request message, if the request message is contained within one packet.

15. The method of claim 1 wherein the request message is a GET request.

16. The method of claim of claim 1 further comprising the step of selecting the proxy cache from a plurality of different proxy caches prior to receiving the at least one packet containing the request message.

17. The method of claim 1 wherein the request message is a POST request.

18. The method of claim 1 wherein the request message is a HEAD request.

19. A proxy redirector for redirecting an HTTP connection request from a client that is directed to an origin server to a proxy cache over a packet-based computer network comprising:

means for receiving at least one packet containing the request message, the a least one packet having an IP header comprising a destination address of an origin server, the request message including a complete address of a specified object at the origin server;

means for modified the request message by combining the destination address of the origin server with the complete address at the packet level; and

means for forwarding the at least one packet containing the modified request message to the proxy cache over the packet-based network;

wherein the proxy cache is a standard proxy cache and the means for forwarding redirects the at least one packet containing the modified request to the standard proxy cache transparently from the standpoints of both the client and the proxy cache, the proxy cache being able to determine whether it can satisfy the request for the specified object by using the combined destination address of the origin server and the complete address contained in the modified request message.

20. The proxy redirector of claim 19 wherein the means for modifying comprises means for prefixing the complete address in the request message with the destination address of the origin server.

21. The proxy redirector of claim 20 further comprising:

means for translating the destination address in an IP header of the at least one packet containing the modified request message to the address of the proxy cache.

22. The proxy redirector of claim 21 further comprising:

means for translating a source address in the IP header of the at least one packet containing the modified request message from an address of the client to an address of the proxy redirector.

23. The proxy redirector of claim 21 further comprising:

means for modifying a value in an acknowledged byte sequence number field in a TCP header in packets received from the proxy cache, commencing with an acknowledgment to the modified request message, to reflect the increased number of bytes in the modified request message resulting from prefixing the complete address with the destination address of the origin server.

24. The proxy redirector of claim 23 wherein the means for modifying the value in the acknowledged byte sequence number field decreases the value in the acknowledged byte sequence number field by the number of bytes added by prefixing the destination address of the origin server to the complete address in the request message.

25. The proxy redirector of claim 24 further comprising:

means for translating in the IP header the destination address to the address of the client and the source address to that of the origin server in the packets received from the proxy cache commencing with the acknowledgment to the request message.

26. The proxy redirector of claim 21 further comprising:

means for modifying a value in a sent byte sequence number field to reflect the increased number of bytes in the modified request message resulting from prefixing the complete address with the destination address of the origin server in a TCP header in packets received from the client following receipt of the request message.

27. The proxy redirector of claim 26 wherein the means for modifying the value in the sent byte sequence number field increases the value in the sent byte sequence number field by the number of bytes added by prefixing the destination address of the origin server to the complete address in the request message.

28. The proxy redirector of claim 27 further comprising:

means for translating the destination address to the address of the proxy cache in the IP header each packet received from the client following receipt of the request message.

29. The proxy redirector of claim 20 further comprising:

means for receiving a packet from the proxy cache indicating a maximum segment size that packets sent to it thereafter should have;

means for reducing by a predetermined number the maximum segment size of packets to be thereafter sent to the proxy cache; and

means for sending a packet to the client indicating the reduced maximum segment size that packets thereafter sent by the client to the origin server should have.

30. The proxy redirector of claim 29 wherein the predetermined number is equal at least to the maximum number of bytes added by the prefix of the destination address to the complete address in the request message.

31. The proxy redirector of claim 30 further comprising:

means for successively shifting to a next packet in the request message overflow bytes caused by the prefix of the destination address to the complete address in the request message, if the request message is contained within more than one packet; and

means for changing a length-of-packet parameter in the IP header in a last packet of a multi-packet request message to reflect the bytes shifted into the last packet from a previous packet.

32. The proxy redirector of claim 30 wherein the proxy cache further comprises means for changing a length-of-packet parameter in the IP header to reflect the change in the length of that packet caused by the prefix of the destination address to the complete address in the request message, if the length of the request message is contained within one packet.

33. The proxy redirector of claim 19 further comprising means for selecting the proxy cache from a plurality of different proxy caches.

34. The proxy redirector of claim 19 wherein the means for modifying the request message is a gateway program dynamically loaded on a programmable network element.

35. The proxy redirector of claim 19 wherein the request message is a GET request.

36. The proxy redirector of claim 19 wherein the request message is a POST request.

37. The proxy redirector of claim 19 wherein the request message is a HEAD request.

38. A computer readable medium storing computer program instructions which are executable on a computer system implementing a Layer 4 switch for redirecting an HTTP connection request from a client to a proxy cache over a packet-based computer network, said computer program instructions comprising instructions defining the steps of:

receiving at least one packet containing the request message, the at least one packet having an IP header comprising a destination address of an origin server, the request message including a complete address of a specified object at the origin server;

modifying, at the packet level, the request message by combining the destination address of the origin server with the complete address; and

forwarding the at least one packet containing the modified request message to the proxy cache over the packet-based network;

wherein the proxy cache is a standard proxy cache and, in the forwarding step, the at least one packet containing the modified request is redirected to the standard proxy cache transparently from the standpoints of both the client and the proxy cache, the proxy cache being able to determine whether it can satisfy the request for the specified object by using the combined destination address of the origin server and the complete address contained in the modified request message.

39. The computer readable memory of claim 38 wherein the step of modifying comprises the step of prefixing the complete address in the request message with the destination address of the origin server.

40. The computer readable memory of claim 39 wherein said computer program instructions further comprise instructions defining the step of:

translating the destination address in an IP header of the at least one packet containing the modified request message to the address of the proxy cache.

41. The computer readable memory of claim 40 wherein said computer program instructions further comprise instructions defining the step of:

translating a source address in the IP header of the at least one packet containing the modified request message from an address of the client to an address of the Layer 4 switch.

42. The computer readable memory of claim 40 wherein said computer program instructions further comprise instructions defining the step of:

in a TCP header in packets received from the proxy cache commencing with an acknowledgment to the modified request message, modifying a value in a acknowledged byte sequence number field to reflect the increased number of bytes in the modified request message resulting from the step of prefixing the complete address with the destination address of the origin server.

43. The computer readable memory of claim 42 wherein the step of modifying the value in the acknowledged byte sequence number field comprises the step of decreasing the value in the acknowledged byte sequence number field by the number of bytes added by prefixing the destination address of the origio server to the complete address in the request message.

44. The computer readable memory of claim 43 wherein said computer program instructions further comprise instructions defining the step of:

translating in the IP header the destination address to the address of the client and the source address to that of the origin server in the packets received from the proxy cache commencing with the acknowledgment to the request message.

45. The computer readable memory of claim 40 wherein said computer program instructions further comprise instructions defining the step of:

modifying a value in a sent byte sequence number field to reflect the increased number of bytes in the modified request message resulting from prefixing the complete address with the destination address of the origin server in a TCP header in packets received from the client following receipt of the request message.

46. The computer readable memory of claim 45 wherein the step of modifying the value in the sent byte sequence number field comprises the step of increasing the value in the sent byte sequence number field by the number of bytes added by prefixing the destination address of the origin server to the complete address in the fequest message.

47. The computer readable memory of claim 46 wherein said computer program instructions further comprise instructions defining the step of:

translating the destination address to the address of the proxy cache in the IP header each packet received from the client following receipt of the request message.

48. The computer readable memory of claim 39 wherein said computer program instructions further comprises instructions defining, prior to the step of receiving the at least one packet containing the request message, the steps of:

receiving a packet from the proxy cache indicating a maximum segment size that packets sent to it thereafter should have;

reducing by a predetermined number the maximum segment size of packets to be thereafter sent to the proxy cache; and

sending a packet to the clieot indicating the reduced maximum segment size that packets thereafter sent by the client to the origin server should have.

49. The computer readable memory of claim 48 wherein the predetermined number is equal at least to the maximum number of bytes added by the step of prefixing the destination address to the complete address in the request message.

50. The computer readable memory of claim 49 wherein said computer program instructions further comprise instructions defining the steps of:

successively shifting to a next packet in the request message overflow bytes caused by the step of prefixing the destination address to the complete address in the request message, if the request message is contained within more than one packet; and

changing a length-of-packet parameter in the IP header in a last packet of a multi-packet request message to reflect the bytes shifted into the last packet from a previous packet.

51. The computer readable memory of claim 49 wherein said computer program instructions further comprise instructions defining the step of changing a length-of-packet parameter in the IP header to reflect the change in the length of that packet caused by the step of prefixing the destination address to the complete address in the request message, if the request message is contained within one packet.

52. The computer readable memory of claim of claim 38 wherein said computer program instructions further comprise instructions defining the step of selecting the proxy cache from a plurality of different proxy caches prior to receiving the at least one packet containing the request message.

53. The computer readable memory of claim 38 wherein the request message is a GET request.

54. The computer readable memory of claim 38 wherein the request message is a POST request.

55. The computer readable memory of claim 38 wherein the request message is a HEAD request.

* * * * *

US006330605B1

(12) United States Patent
Christensen et al.

(10) Patent No.: US 6,330,605 B1
(45) Date of Patent: Dec. 11, 2001

(54) **PROXY CACHE CLUSTER**

(75) Inventors: **Brent Ray Christensen**, Cedar Hills; **Howard Rollin Davis**, Salem; **Adam Lee Harral**, Pleasant Grove, all of UT (US)

(73) Assignee: **Volera, Inc.**, San Jose, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/195,982**

(22) Filed: **Nov. 19, 1998**

(51) **Int. Cl.**[7] **G06F 15/16**
(52) **U.S. Cl.** **709/226**
(58) **Field of Search** 700/200, 203, 700/226, 233, 234, 235, 238, 239, 240, 241

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,151,688	9/1992	Wipfel et al.	345/104
5,495,426	2/1996	Waclawsky et al.	365/514
5,642,511	6/1997	Chow et al.	395/701
5,784,566	7/1998	Vivan et al.	709/229
5,924,116	7/1999	Aggarwal et al.	711/122
5,950,205	9/1999	Wilford et al.	370/392
5,961,593	10/1999	Gabber et al.	709/219
5,964,891	10/1999	Caswell et al.	714/31
5,999,734 *	12/1999	Willis et al.	709/200
6,014,677	1/2000	Jenkins et al.	707/501
6,018,619	1/2000	Allard et al.	709/224
6,094,708	7/2000	Hilla et al.	711/138
6,111,877	8/2000	Wilford et al.	370/392
6,112,228	8/2000	Earl et al.	709/205
6,178,160	1/2001	Bolton et al.	370/255
6,185,598 *	2/2001	Farber et al.	709/200
6,185,625 *	2/2001	Tso et al.	709/247
6,230,241	5/2001	McKenney	711/118
6,240,461	5/2001	Cieslak et al.	709/235

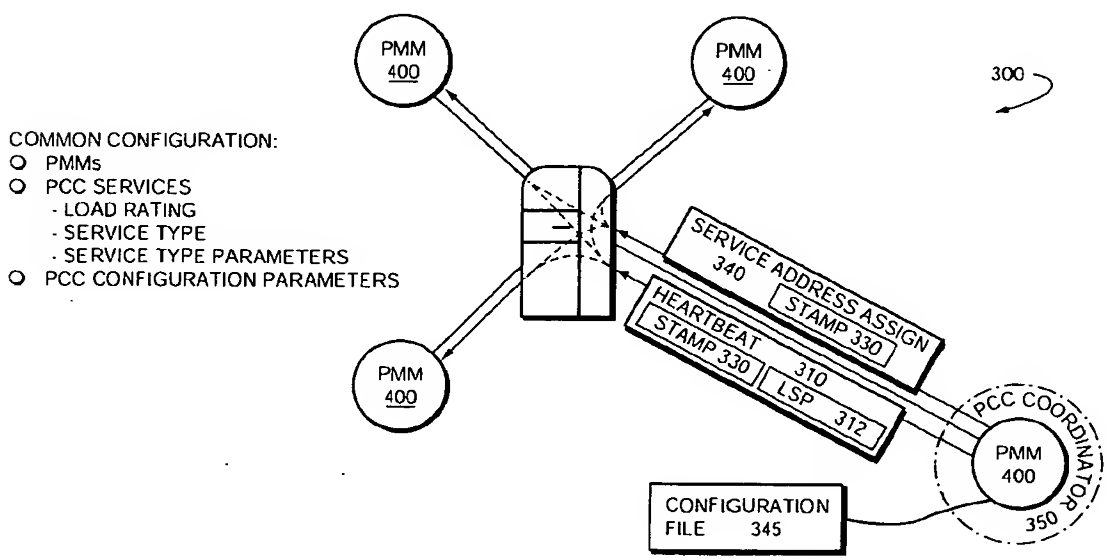
* cited by examiner

Primary Examiner—David Y. Eng
(74) *Attorney, Agent, or Firm*—Cesari and McKenna, LLP

(57) **ABSTRACT**

A proxy cache cluster (PCC) couples to a service provider of a communications network to increase the availability of services offered by the provider to clients connected to the network. The clients access the services by issuing requests to network addresses associated with these services. The PCC increases the availability of the services by receiving and servicing those requests on behalf of the service provider in accordance with a proxy cache clustering technique.

29 Claims, 5 Drawing Sheets



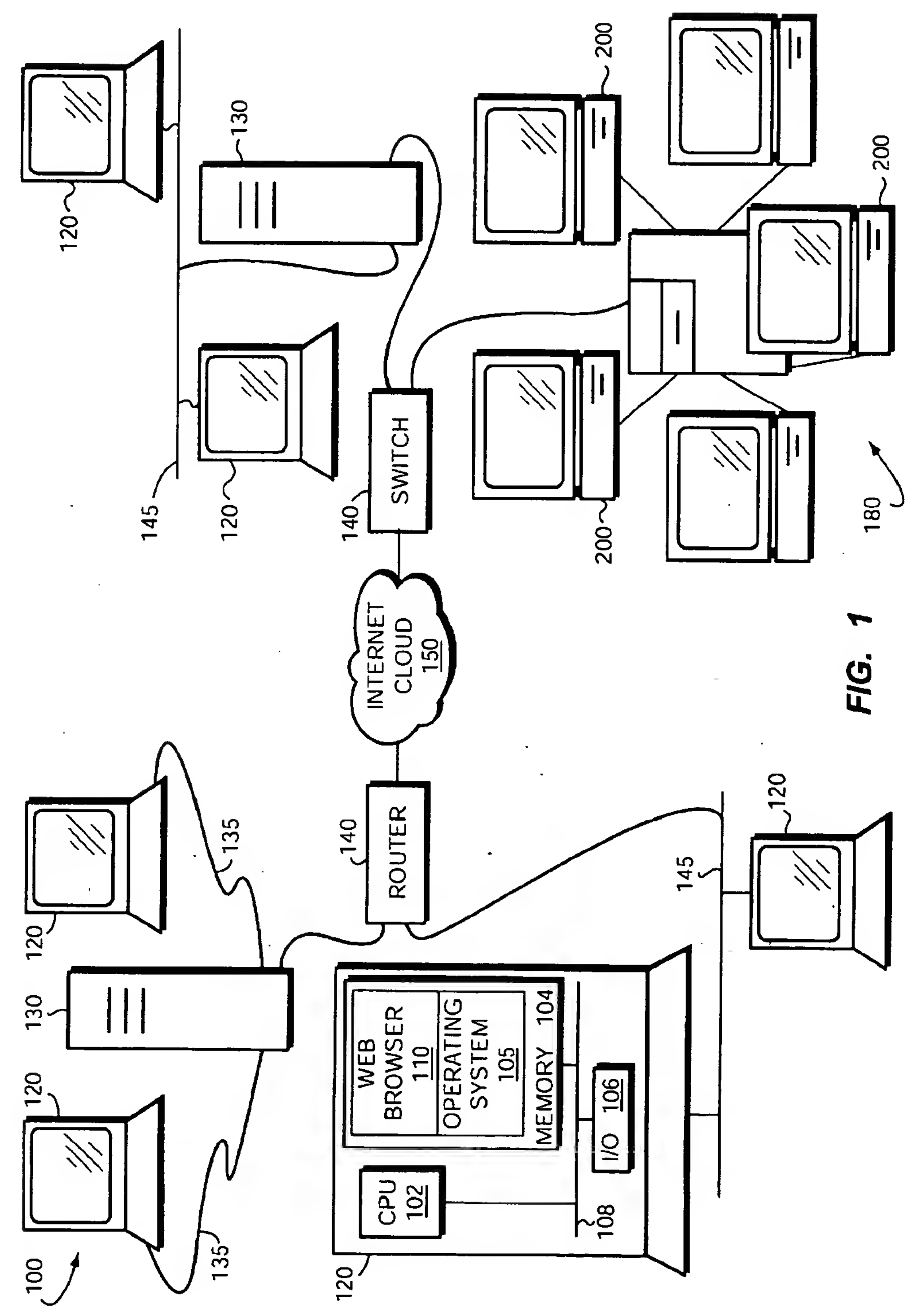

100
120
130
135
140
ROUTER
INTERNET
CLOUD
150
SWITCH
145
WEB
BROWSER
110
OPERATING
SYSTEM
105
MEMORY 104
CPU
102
I/O 106
108
180
200

FIG. 1

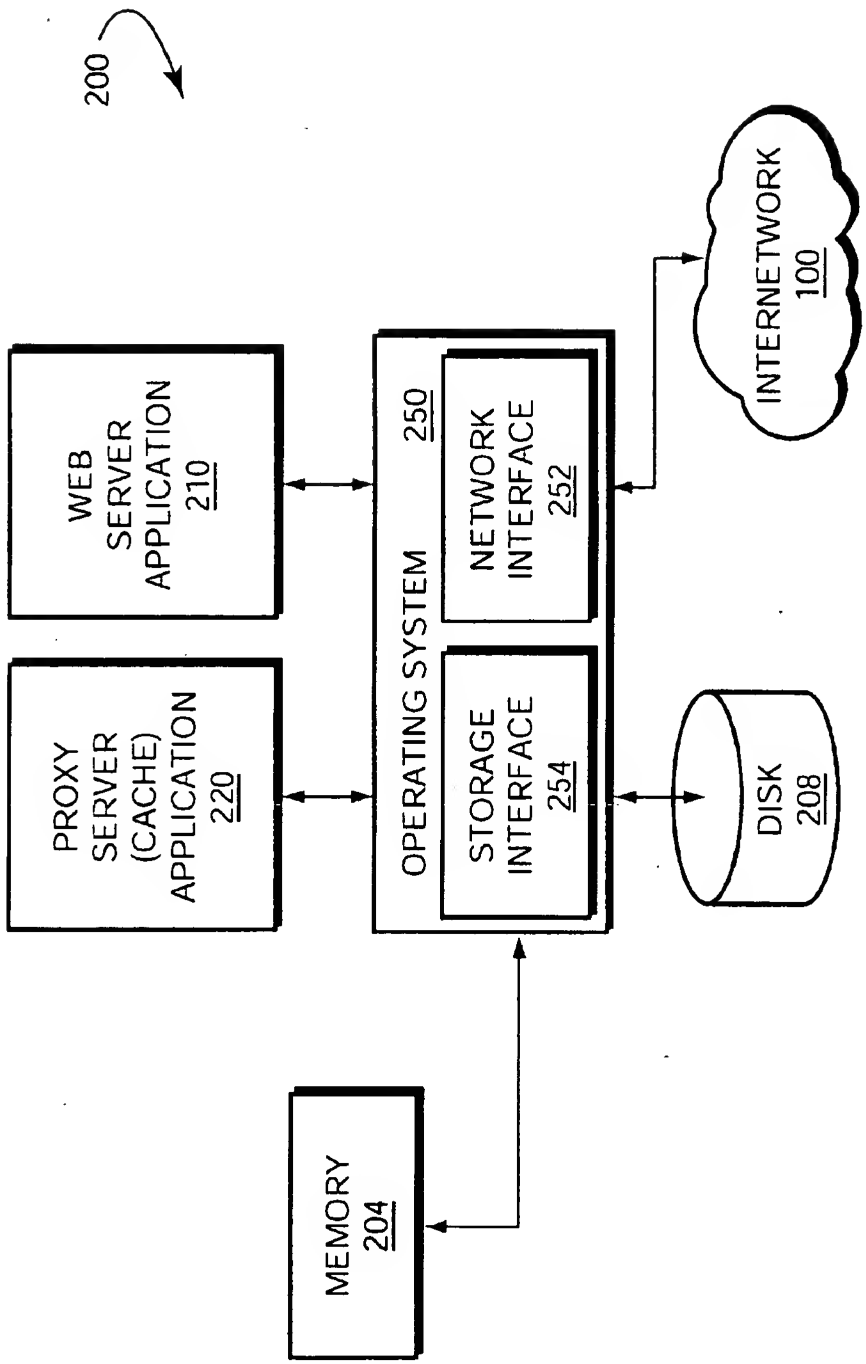

200
WEB SERVER APPLICATION 210
PROXY SERVER (CACHE) APPLICATION 220
OPERATING SYSTEM 250
NETWORK INTERFACE 252
STORAGE INTERFACE 254
INTERNETWORK 100
DISK 208
MEMORY 204

FIG. 2

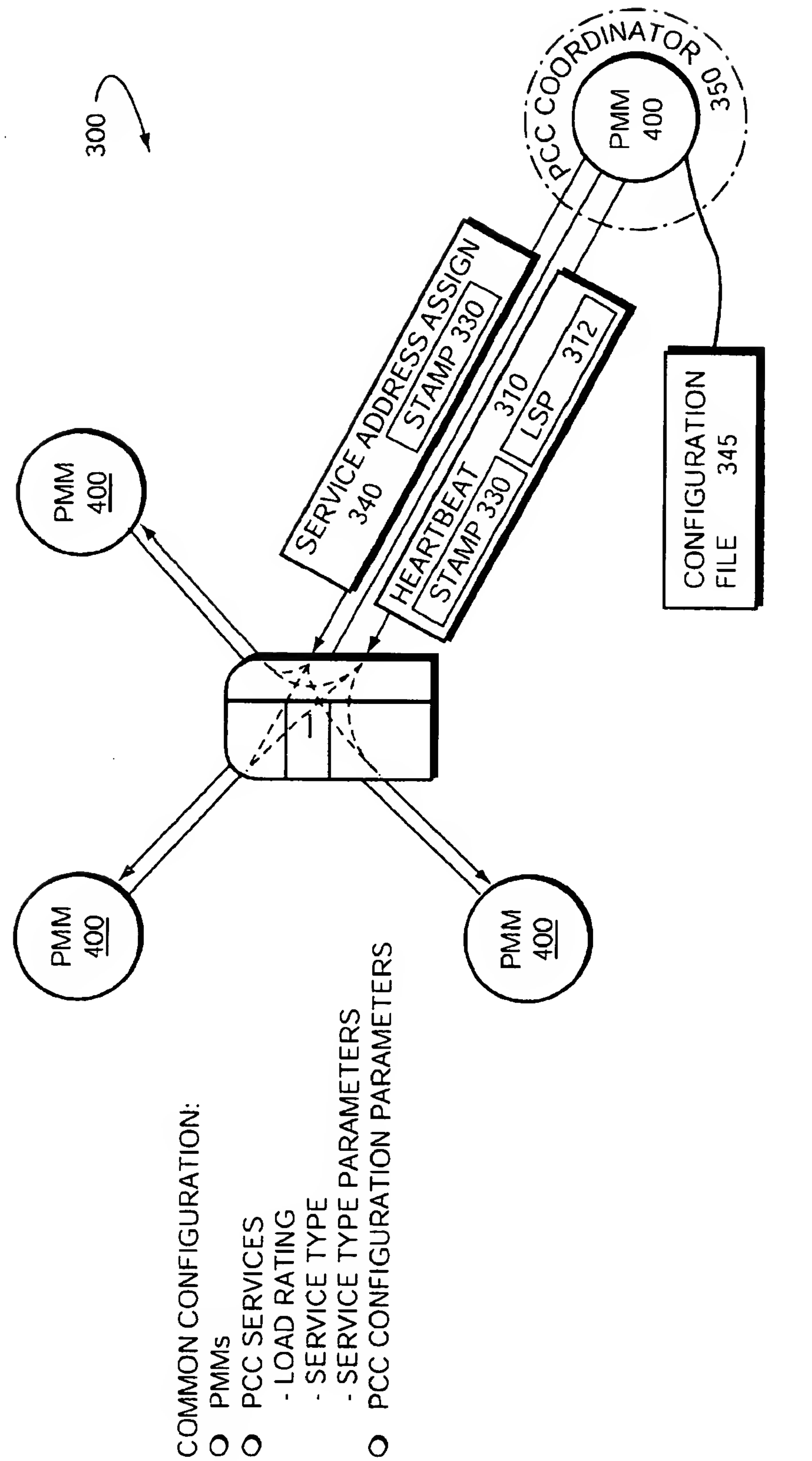

300
PCC COORDINATOR 350
PMM 400
SERVICE ADDRESS ASSIGN 340
STAMP 330
HEARTBEAT 310
STAMP 330
LSP 312
CONFIGURATION FILE 345
PMM 400
PMM 400
PMM 400
1
COMMON CONFIGURATION:
O PMMs
O PCC SERVICES
- LOAD RATING
- SERVICE TYPE
- SERVICE TYPE PARAMETERS
O PCC CONFIGURATION PARAMETERS

FIG. 3

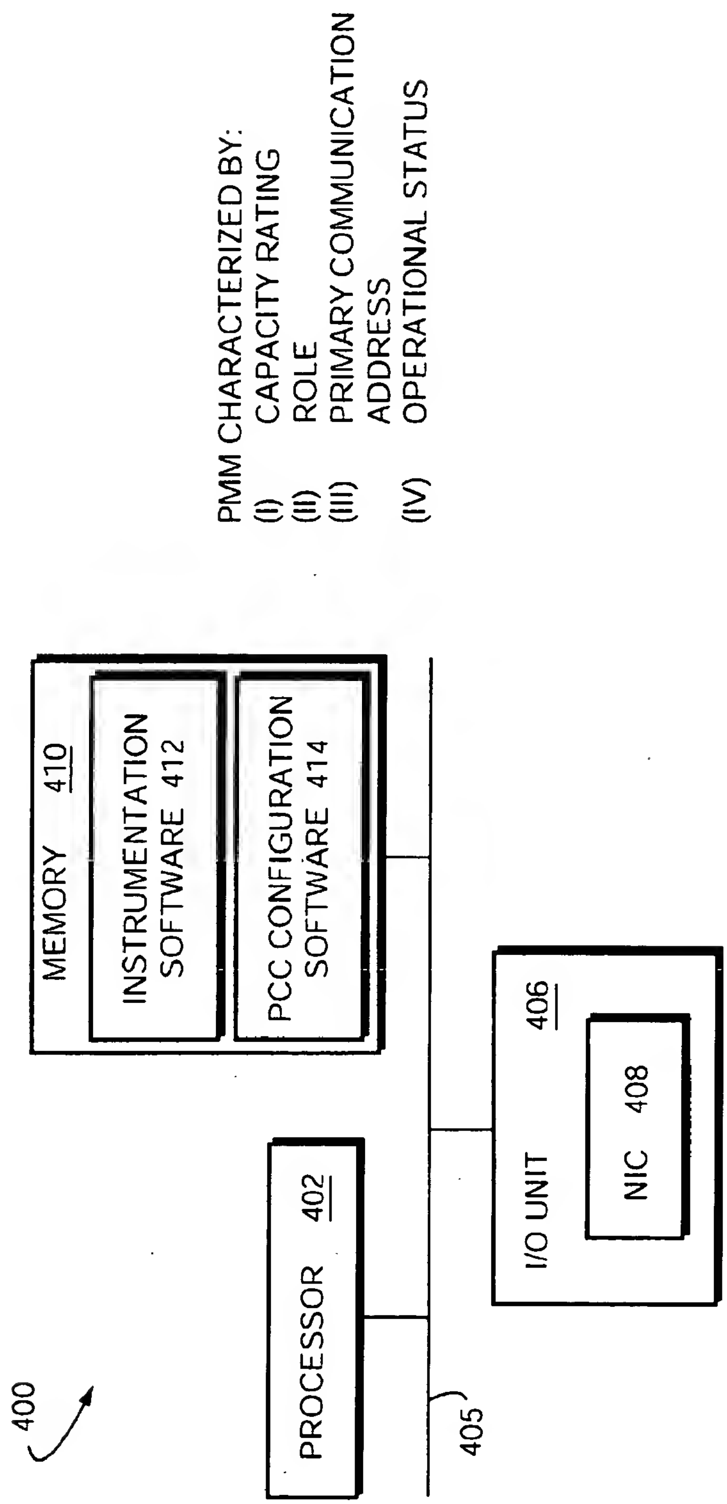

PMM CHARACTERIZED BY:
(I) CAPACITY RATING
(II) ROLE
(III) PRIMARY COMMUNICATION ADDRESS
(IV) OPERATIONAL STATUS
MEMORY 410
INSTRUMENTATION SOFTWARE 412
PCC CONFIGURATION SOFTWARE 414
PROCESSOR 402
I/O UNIT 406
NIC 408
400
405

FIG. 4

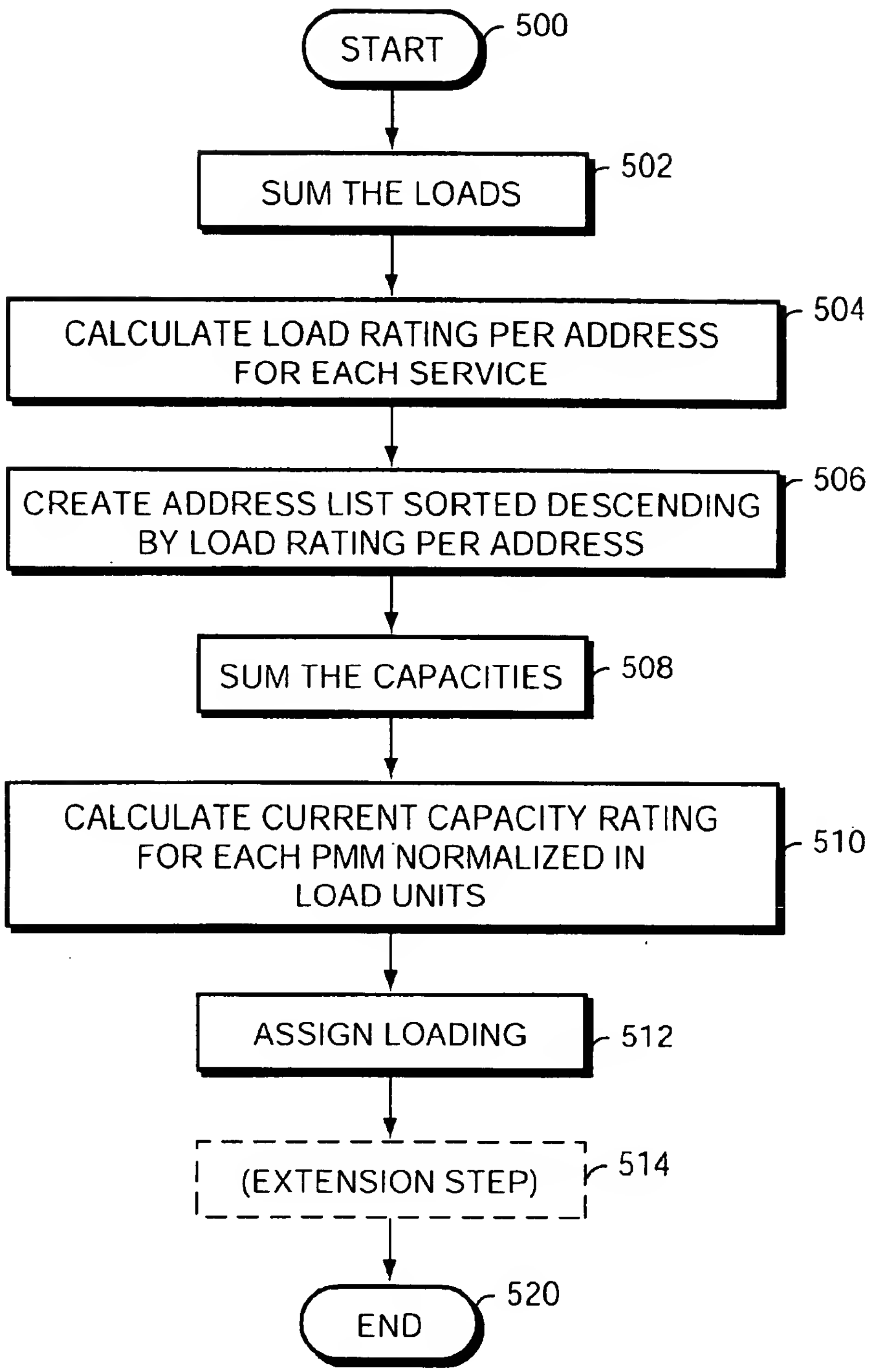

START
500
SUM THE LOADS
502
CALCULATE LOAD RATING PER ADDRESS FOR EACH SERVICE
504
CREATE ADDRESS LIST SORTED DESCENDING BY LOAD RATING PER ADDRESS
506
SUM THE CAPACITIES
508
CALCULATE CURRENT CAPACITY RATING FOR EACH PMM NORMALIZED IN LOAD UNITS
510
ASSIGN LOADING
512
(EXTENSION STEP)
514
END
520

FIG. 5

PROXY CACHE CLUSTER

CROSS-REFERENCE TO RELATED APPLICATION

This invention is related to the following copending and commonly assigned U.S. patent application Ser. No. 09/023,895 titled, Client Inherited Functionality Derived from a Proxy Topology where each Proxy is Independently Configured by Douglas G. Earl et al, filed on Feb. 13, 1998.

FIELD OF THE INVENTION

The present invention relates to services of communications networks and, more specifically, to a system and method for increasing the availability of services offered by a service provider of a communications network.

BACKGROUND OF THE INVENTION

It is increasingly common for users having standalone computers, or computers interconnected by an institutional intranet or local area network, to gain access to various remote sites (such as those on the "World Wide Web") via the well-known Internet communications network. Using resident web browser applications executing on the computers, these clients may navigate among services ("pages") stored on various servers of a service provider ("web site") and may further request these services as desired. In a basic network communication arrangement, clients are free to access any remote web site for which uniform resource locators (URLs) are available.

It is also increasingly common in network applications to provide the web site servers with associated proxy cache servers that link ("front-end") the servers with the Internet. A proxy cache server ("proxy") may be used to accelerate client access to the Internet ("forward proxy") or to accelerate Internet access to a web server ("reverse proxy"). As for the latter reverse proxy environment, the proxy may access frequently requested services from the web servers and store ("host") them locally to effectively speed-up access to future requests for the services. For instance, a proxy may host frequently requested web pages of a web site. In response to a request from a browser executing on a client, the proxy attempts to fulfill that request from its local storage; if it cannot, the proxy forwards the request to a web site server that can satisfy the request. The web server then responds by transferring a stream of information to the proxy, which stores and forwards the information over the Internet onto the client. The illustrative embodiment of the invention described herein is applicable to a reverse proxy environment.

As Internet traffic to the web site increases, the network infrastructure of the service provider may become strained attempting to keep up with the increased traffic. In order to satisfy such demand, the service provider may increase the number of network addresses for a particular service by providing additional web servers and/or associated proxies. These network addresses are typically Tranmission Control Protocol/Internet Protocol (TCP/IP) addresses that are represented by URLs or wordtext (domain) names and that are published in a directory service, such as the well-known Domain Name System (DNS). Computers referred to as name servers implement DNS by mapping between the domain names and TCP/IP address(es).

Since the proxies "front-end" the web servers (and may, in fact, be resident on the web servers) in a reverse proxy environment, the network addresses of the proxies (rather than the actual web site) are generally mapped to the domain name of the service provider. As a result, communication exchanges with the proxies generally comprise IP packets or UDP/TCP-socketed traffic, such as socket requests and responses. A socket is essentially an interface between an application layer and transport layer of a protocol stack that enables the transport layer to identify which application it must communicate with in the application layer. For example, a socket interfaces to a TCP/IP protocol stack via a set of application programming interfaces (API) consisting of a plurality of entry points into that stack. Applications that require TCP/IP connectivity typically utilize the socket API to interface into the TCP/IP stack.

For a connection-oriented protocol such as TCP, the socket may be considered a session; however, for a connectionless protocol such as IP datagram using the User Datagram Protocol (UDP), the socket is an entity/handle that the networking software (protocol stack) uses to uniquely identify an application layer end point, typically through the use of port numbers. The software entity within the server that manages the communication exchanges is a TCP/IP process, which is schematically illustrated as layers of a typical Internet communications protocol stack. Protocol stacks and the TCP/IP reference model are well-known and are, for example, described in *Computer Networks* by Andrew S. Tanenbaum, printed by Prentice Hall PTR, Upper Saddle River, N.J., 1996.

Thus to access a particular service, a client issues a request to the domain name of the service. A name server receives the request, looks up the domain name and returns all of the mapped (proxy) associated network addresses to the client. The client chooses a first of the addresses directed to a first proxy and sends a service request to that proxy. If the proxy has failed and is "down", the request is ignored; after waiting a period of time, the client may issue another request to that proxy. After sending one or more non-responsive requests to that address, the client may issue a request directed to a second address of a second proxy associated with the service. This time the request may be received and serviced by the second proxy.

Subsequent requests to the particular service may be satisfied as the client selects each of the remaining addresses in, e.g., a round-robin manner, until it comes back to the first address, where a request may again be directed to the first proxy. As long as this proxy is down, though, requests directed to that first network address are ignored. In addition to frustrating the client, such non-responsive requests may prove costly to the service provider in terms of lost revenue and service down-time. As for the latter, adding a new service or expanding an old service of, e.g., a web server may be time consuming and fraught with configuration challenges that typically result in lengthly installations. The present invention is directed to alleviating such time consuming activities and, more specifically, to efficiently increasing the availability of services offered by a service provider.

SUMMARY OF THE INVENTION

The present invention relates to a proxy cache cluster (PCC) coupled to a service provider of a communications network to increase the availability of services offered by the provider to clients connected to the network. As noted, the clients access the services by issuing requests to network addresses associated with these services. The PCC increases the availability of the services by receiving and servicing those requests on behalf of the service provider in accor-

dance with a novel proxy cache clustering technique described herein.

Specifically, the PCC comprises a group of processor/memory mechanisms (PMMs) that cooperately interact as a system to host services associated with the network addresses. By functioning as a system of proxy cache servers defined by a common configuration of PMMs, hosted PCC services and static PCC configuration parameters, the PCC improves availability, performance and scalability of the service provider, which preferably cnmprises a plurality of web servers. Scalability of the service provider may is be further achieved by adding PMMs, PCC service addresses and/or PMM network addresses to the PCC.

Operatinnally, a designated PMM of the PCC functions as a coordinator to administer the commnn configuration by, in part, implementing a load balancing aspect of the clustering technique to substantially optimize assignment of the network addresses to active PMM members. Load balancing is preferably implemented using characteristics of each PMM and PCC service, such as PMM capacity, PCC service load and PCC service address(es). Once the PCC is "balanced", each PMM attends tn client requests directed to its assigned address(es). In the event of a failure to one of their members, the active PMMs collaborate with the PCC coordinator to reassign the failed PMM's service addresses among the remaining active members.

Advantageously, the inventinn facilitates the addition and/or expansion of services offered by a service provider by dynamically balancing service loads among the existing PMM members of the PCC. That is, the proxy cache clustering technique enables load balancing nf the services using the service addresses assigned to the PMM members rather than employing additinnal web servers, as in the prior art. Moreover, the inventive clustering technique further increases the availability of thnse services by providing dynamic fault detectinn/failover recovery in the event of failure to the PMMs.

BRIEF DESCRIPTION OF THE DRAWINGS

The above and further advantages of the invention may be better understood by referring to the following description in conjunction with the accompanying drawings in which like reference numerals indicate identical or functionally similar elements:

FIG. 1 is a block diagram of a computer internetwork including a collection of network segments connected to a plurality of client and server computers, the latter of which may be organized as a service provider;

FIG. 2 is a highly schematized diagram of software components of the service provider server of FIG. 1;

FIG. 3 is a schematic block diagram of an inventive proxy cache cluster (PCC) comprising a group of processor/memory mechanisms (PMMs) that cooperately interact to host PCC services associated with network addresses of the service provider;

FIG. 4 is a schematic block diagram of a PMM of the present invention; and

FIG. 5 is a flowchart depicting a sequence of steps associated with a load balancing technique according to the present invention.

DETAILED DESCRIPTION OF AN ILLUSTRATIVE EMBODIMENT

FIG. 1 is a schematic block diagram of a computer internetwork 100 comprising a collection of network segments connected to a plurality of computers 120, 130, 140 and 200. Each computer generally comprises a central processing unit (CPU) 102, a memory 104 and an input/output (I/O) unit 106 interconnected by a system hus 108. The memory 104 may comprise stnrage locations typically composed of random access memory (RAM) devices, which are addressable by the CPU 102 and I/O unit 106. An operating system 105, portions of which are typically resident in memory and executed by CPU, functionally organizes the computer by, inter alia, invoking network operations in support of application programs executing on the CPU. An example of such an application program is a web browser 110, such as Netscape Navigator™ available from Netscape Communicatinns, Inc.

The network segments may comprise lncal area networks 145 or intranets, point-to-point links 135 and an Internet cloud 150. Collectively, the segments are interconnected by intermediate stations 140, such as a network switch or router, and cnnfigured to form an internetwnrk of computers that communicate by exchanging data packets according to a predefined set of protocols, such as the Transmissinn Control Protocol/Internet Protocnl (TCP/IP). It should be noted that other techniques/protncnls, such as Internet Packet Exchange (IPX) protocol and the Hypertext Transfer Protocol (HTTP), may be advantageously used with the present invention.

In the illustrative embodiment, the internetwnrk 100 is organized in accordance with a client/server architecture wherein computers 120 are personal computers or workstatinns configured as clients for interaction with users and computers 130, 200 are configured as servers that perform services as directed by the clients. Fnr example, the servers 200 may be configured to nperate as a service provider (e.g., a web site 180) as described further herein, whereas servers 130 may be configured as domain name system (DNS) servers and/or Internet provider servers. In general, the DNS servers provide the clients 120 with the network (e.g., IP) address(es) of requested services in response to packets directed to the domain names for those services. The Internet providers, on the other hand, provide Internet access to the clients via, e.g., dial-up telephone lines or cable links.

The client 120 may utilize the web browser 110 to gain access to the web site 180 and to navigate, view or retrieve services stored on the servers 200, hereinafter "web" servers. In order to effectively speed-up access to the service provider and reduce the retrieval time for stored services, each web server 200 may be further provided with a proxy cache server. FIG. 2 is a highly schematized diagram of software components of the web server 200 including an operating system 250 having utility programs that interact with various application program components to provide, e.g., a storage interface 254 and a network interface 252, the latter enabling communication with a client browser 110 over the internetwork 100. The application program components include a web server application 210 and a proxy server application ("proxy") 220.

As noted, the proxy 220 "front-ends" the web server such that the network address of the proxy (rather than the actual web site) is published in the DNS server 130 and mapped to the domain name of the service provider. To access a service of the service provider 180, the client sends a request packet directed to the network address of a particular proxy 220 of the web site. The proxy 220 receives the request from the browser 110 and, if the client is authorized to access services from the web site, the proxy attempts to fulfill that request locally from information stored, e.g., in memory 204 or on disk 208; in either case, the memory and/or disk function as

a "cache" for quickly storing and retrieving the services. If it cannot satisfy the request, the proxy forwards the request onto its web server application **210**. The web server application then responds by transferring a stream of information to the proxy, which stores and forwards the information onto the browser **110**. Although the proxy **220** is shown as resident on the web server **200**, it should be noted that the proxy may also be configured to run on a separate server platform.

A problem arises with the arrangement described above when the particular proxy **220** is "down" because, e.g., of a failure with the server. After repeated attempts to access the proxy are ignored, the client may send a service request directed to a network address of another proxy (and another web server **200**) of the web site **180**; although this request may be serviced by the latter proxy and/or web server application **210**, the previous non-responsive requests may prove costly to the service provider in terms of lost revenue, service down-time and user frustration. The present invention is directed, in part, to solving this problem. Moreover, adding a new service or expanding an old service of the web site may be accompanied by configuration issues that result in time consuming installations. The present invention is further directed to alleviating such time consuming activities.

In accordance with the present invention, a proxy cache cluster (PCC) "front-ends" the servers of a service provider to increase the availability of services offered by the provider. As noted, the clients access the services by issuing requests to network addresses associated with the services. The PCC increases the availability of the services by receiving and servicing those requests on behalf of the service provider in accordance with a novel proxy cache clustering technique described herein. FIG. **3** is a schematic block diagram of a PCC **300** comprising a cluster of processor/memory mechanisms (PMMs **400**) with associated network connectivity that share a common configuration. The PMMs cooperately interact as a system to host PCC services associated with network addresses of the service provider **180**. In fact, the common configuration describes the PCC in terms of the PMMs, static PCC configuration parameters and hosted PCC services, which may include any service that is provided on the World Wide Web; an example of a common web site service is an HTTP service.

According to an aspect of the invention, a PCC service is characterized by (i) a load rating, which is a number/value that reflects a measure of the PCC service's resource source consumption, such as the amount of traffic at the web site **180**. Notably, the actual value is not specified; just a consistent value to measure the resource consumption metric. Furthermore, the value is a relative value; i.e., relative to load rating of the other services hosted by the PCC. Measurement of this metric can be performed manually or via a software agent that dynamically (event-driven or continuous) assesses the load rating. The agent may comprise conventional instrumentation software, such as a simple network management protocol (SNMP) agent, or any other application that instruments the service to calculate its rating. Calculations are performed using normalized units of measurement to provide flexibility in the ratings and to facilitate dynamic ("on-the-fly") computer-generated measurements, particularly in the event of inclusion of additional PMMs to a cluster in accordance with the novel clustering technique. The agent further maintains (updates) these generated statistics for use when balancing the hosted services across the PCC.

The PCC service is further characterized by (ii) service type, such as an HTTP proxy service, an HTTP accelerator service or a file transfer protocol (FTP) proxy service; further examples of service types include Real Audio, Real Video, NNTP and DNS; and (iii) service type parameters that are unique to each service. Typical examples of conventional parameters run by an HTTP proxy service include (a) a list of network addresses of, e.g., the web site that allows access to the web servers, (b) whether logging is activated, (c) the format of the activated log (common or extended) and (d) the log roll rate. Most web sites that provide an HTTP service run logging operations to determine the type of requests issued by users, the kinds of errors received by those users and the source network addresses of the requests. This latter log provides an indication of geography with respect to, e.g., the locations of the highest concentration of users.

The PMMs are organized as a PCC in accordance with the proxy cache clustering technique that dynamically assigns each PMM to the PCC. To that end, each PMM is configured with a unique identifier (ID), a network address and PCC configuration software to enable participation in the clustering process. The unique ID may be the media access control (MAC) address of a network interface card of the PMM or it may be the network address of the PMM. Once configured and activated, the PMM "listens" for a mechanism notifying the PMM that it is a member of a PCC **300**. According to the clustering technique, the notification mechanism is a heartbeat message **310** that is broadcasted to each member of the PCC. In the illustrative embodiment, the heartbeat message **310** is effectively an "I'm (still) alive" message that informs the cluster members that the sender is a member of the cluster and (still) available. The heartbeat message may include a conventional link state packet (LSP) mechanism **312** that allows dynamic cluster management. As the PMM receives heartbeat messages and "learns" about other PMMs, it sends its own heartbeat message **310** to those learned PMMs.

A designated PMM of the PCC functions as a PCC coordinator **350** to administer the common configuration and, as such, is responsible for assigning PCC service address(es) to each PMM. These service address assignments are contained in a message **340** that is periodically distributed to the PMMs **400** by the coordinator; each PMM is thus aware of all its peer PMMs in the cluster. The service address assignments **340** are derived from information contained in a configuration file **345**. Each message transmitted by a member of the PCC, including the heartbeat **310** and service address assignments **340**, is identified by a PCC Identifier or stamp **330**. The stamp **330** is unique to the PCC and PCC configuration, and may be manifested as information appended to the body of a packet, or contained within an extended header or footer of the packet. All messages issued by the PCC coordinator are also marked with the stamp **330** so that the recipient PMMs can be assured ("trust") that the received messages are from their coordinator. In an alternate embodiment, the stamp may be a digital signature when utilizing a public key cryptography system.

FIG. **4** is a schematic block diagram of a PMM **400**. In the illustrative embodiment, the PMM is a personal computer or workstation configured as a server according to a single processor, single memory (SPSM) model, although in an alternate embodiment, the PMM may comprise a multiprocessor server with or without multi-memory. In any event, the PMM contains processing (processor **402**), memory **410** and input/output (I/O **406**) resources having characteristics that differ relative to other PMMs of the PCC, but that cooperate to provide the inventive proxy cache clustering functions.

According to the invention, a PMM is characterized by (i) capacity rating, which is number/value that reflects a measure of the PMM's ability to provide resources. As with the service loading rating, the value is relative to the capacity rating of the other PMMs in the PCC 300 and measurement of this metric can be done manually or via conventional instrumentation software 412 (e.g., SNMP agents). Factors influencing the capacity rating of a PMM 400 include the "size" of its network connection, the "amount" of memory and the "speed" of its processor. For example, a PMM may have faster I/O capability than other PMMs due to its high-performance network interface card (NIC 408) "pipeline" (e.g., 100 base T or Sonet) to heterogeneous network segment media (e.g., 10 Mb, 100 Mb and/or Gigabit Ethernet or FDDI). Another PMM may provide faster processing capabilites than other PMMs because it contains a high-performance processor 402. The relative resource characteristics of the PMMs are taken into account by the inventive clustering technique to produce, inter alia, the capacity rating.

The PMM may also be characterized by (ii) role. The PMM role describes various activity states of the PMM as manifested in the configuration file 340 and/or by the heartbeat 310. For example, an active role allows the PMM to participate in all PCC activities, whereas a standby role prohibits assignments to the PMM until at least one other PMM is down or unusable. It should be noted that a PMM may be a member of more than one PCC; e.g., there may be 2 PMMs on a low-usage service cluster and 4 PMMs on another high-usage service cluster. A PMM of the low-usage cluster may be configured into the high-usage cluster as a standby which means the PMM will continue operating as a member of the low-usage cluster until a failure occurs in the high-usage cluster. At that time, the PMM will assume additional responsibilities for that latter cluster. The capacity rating of the standby PMM determines whether the machine may be utilized in such a role, which effectively sacrifices performance of the slower, low-usage service to ensure performance of the higher-usage service.

In the case of the standby role, the PMM 400 transmits a heartbeat message 310 to inform the high-usage PCC coordinator of its availability in the event of a problem. The configuration file 340 specifies the roles of the member PMMs and their network addresses. Thus if a failure occurs, the PCC coordinator 350 sends a message to the standby PMM notifying it that it is no longer on standby, that it is an active member of the high-usage cluster and as to its address assignment(s). The configuration file is updated to reflect this changed role status and is distributed to all PMM members of the PCC. If the errant PMM comes back online, the PCC coordinator 350 transitions the PMM 400 back to the standby mode and rebalances the cluster with the errant PMM. The standby PMM finishes its current and pending work, and reverts to standby mode. It should be noted that multiple PMMs may assume the standby role for a PCC and that a PMM may be a member of multiple clusters. Moreover, a PMM may assume standby status on some clusters and active status on others. A PMM that assumes such a dual role distinguishes its responsibilities and communication exchanges among the various PCCs by way of the PCC stamp 330.

An unusable role indicates that the PMM cannot be used at all by the PCC, while the down role is automatically assigned by the PCC coordinator upon failure to detect a heartbeat within the prescribed period. In particular, the unusable role state is used when reconfiguring or reloading a PMM with, e.g., new software (upgrade the clustering software or protocol stack). The PCC coordinator 350 marks the PMM unusable and rebalances the cluster. During this rebalancing period, the PMM 400 finishes its work on requests pending in an incoming queue and then shuts itself down (from the point of view of being a cluster member). Once the PMM is reloaded, it transitions into the active state and, upon other members detecting its presence, the PMM rejoins the cluster by receiving assignments from the PCC coordinator.

Note that an "administrative fault" can be created by changing any PMM's role status to unusable. For example, assume a 3-PMM cluster is moved from one location to another facility. Prior to moving the PCC, a first PMM is placed in the unusable state and proceeds to empty its incoming queue with pending requests, power down, move to the new facility, and power up. According to the inventive clustering technique, the PCC coordinator immediately performs the tasks of a fault-failover, configuring the PMM out of the PCC until it is relocated and brought online. At that point, the PMM is placed in the active state to rejoin the cluster. The same procedure takes place for a second PMM. However when the last PMM, which is the PCC coordinator, is placed in the unusable role and powered down, the remaining two PMMs elect a new coordinator and the relocating procedure described above takes place. The PMM conveys its role status to the cluster members through the heartbeat message 310; e.g., presence of a heartbeat=active or standby roles, whereas absence of a heartbeat=unusable or down roles.

The PMM 400 may be further characterized by (iii) primary communication address and (iv) operational status. The primary communication address is the network address on which PCC-specific communication among PMMs and PCC coordinator occurs; this address may be used as the unique ID of the PMM. The primary network address is also the identifier that is preferably used to elect a PCC coordinator, e.g., the PMM with lowest IP address. It will be understood to those skilled in the art, however, that other techniques could be used to elect a coordinator, such as use of a MAC address or anything that allows unique identification of the PMM machine.

The operational status of a PMM includes joining, up, down and leaving states. A PMM assumes a joining status when attempting to rejoin its cluster after being offline. Once the PCC coordinator accepts the PMM and rebalances the cluster, the PMM assumes an up status. When the PMM is removed from the cluster, it enters a leaving status to gracefully bring itself to a down status.

Table 1 illustrates an exemplary PCC configuration file 345 containing a list of PMMs (Servers 1–3) by network (IP) address and a list of PCC services (Services 1–2). The configuration file also contains a list of static PCC configuration parameters and specific numbers/values associated with each parameter.

TABLE 1

PCC configuration file
[Main]
name=TestCluster
heartBeatTimerTicks=20
alertTimerTicks=10
joinTimeoutTicks=100
failureSuspectedTimeTicks=50
failureConfirmedTimeTicks=100
countServers=3
countServices=2
[Service 1] ; Forward Proxy
serviceType=1 ; 1 = HTTP Proxy, 2 = HTTP Accel
serviceName=TestCluster.Proxy.Com
serviceNameAndAddressList=137.65.24.210,137.65.24.211
port=8080

TABLE 1-continued

```
PCC configuration file

loadWeight=1
webServerPort=0  ; for accelerators only
webServerNameAndAddressList=  ; for accelerators only
commonLogEnabled=1      ; 0 = no, 1 = yes
extendedLogEnabled=1    ; 0 = no, 1 = yes
indexedLogEnabled=0     ; 0 = no, 1 = yes
[Service 2]    ; Web Server Accelerator of www.test.com
serviceType=2     ; 1 = HTTP proxy, 2 = HTTP accel
port=80
loadWeight=1
serviceName=www.test.com
serviceNameAndAddressList=137.65.24.220,137.65.24.221
webServerPort=80
webServerNameAndAddressList=www.test.com
commonLogEnabled=0      ; 0 = no, 1 = yes
extendedLogEnabled=0    ; 0 = no, 1 = yes
indexedLogEnabled=0     ; 0 = no, 1 = yes
[Server 1]
serverName=Calypso
primaryIPAddress=137.65.24.98
role=1
capacityWeight=1
[Server 2]
serverName=Sapphire
primaryIPAddress=137.65.24.114
role=1
capacityWeight=1
[Server 3]
serverName=Melout
primaryIPAddress=137.65.24.81
role=2
capacityWeight=1
```

As listed in Table 1, the static configuration parameters include (1) failure suspected threshold, (2) failure confirmed threshold, (3) join timeout interval, (4) heartbeat timer interval and (5) alert timer interval. These parameters, which are described below, provide the basis of a fault-failover aspect of the inventive clustering technique.

Failure suspected threshold. As noted, each PMM **400** constantly monitors the heartbeat **310** of other PMM members of the PCC **300**. If a heartbeat message from a particular PMM is not detected by another PMM before this threshold is exceeded, then failure of the particular PMM is suspected. At this time, each PMM actively tries to reestablish communication with the suspected PMM. Note that the LSP portion **312** of the heartbeat message **310** enables each PMM to maintain state information about the other PMM members. For example, whenever a PMM detects a heartbeat of another PMM member (via a recent time stamped heartbeat message from that member), it sets a counter indicating such heartbeat detection. The PMM periodically scans these states and if a heartbeat is not detected within the prescribed period, i.e., the failure suspect threshold, then the PMM transitions to a more aggressive state to confirm the suspected failure. That is, the PMM tries to actively reestablish communication by sending messages to the suspect PMM inquiring as to its status.

Failure confirmed threshold. If attempts by PMM members to actively communicate with a suspect PMM do not succeed within this threshold, failure of the PMM is confirmed, the suspect PMM is effectively eliminated from the PCC and the PCC coordinator **350** begins redefining the loads of the remaining PCC members. Despite the reason for not responding, e.g., the PMM **400** is down, its connecting link is down or a port of the router into which the link is plugged is down, the suspect PMM is no longer capable of functioning as a member of the cluster and no longer has

assignments from the PCC **300**. Further, the PCC will not issue additional assignments to the suspect PMM until its members receive heartbeat messages from the suspect machine; at that time, the PCC coordinator reorganizes the cluster to include the now reappearing PMM.

If the suspect PMM is the current coordinator of the PCC, the cluster members proceed to determine the identity of a new PCC coordinator. A PMM may "volunteer" to be the coordinator and, if others agree, it is elected as such and the PCC is formed. Because the configuration file **340** is readily available to each PMM member, the election process may be quickly effected by examining the file's content and determining which remaining PMM meets the election criteria. In the illustrative embodiment, the PCC coordinator **350** is preferably the PMM **400** having the lowest IP address, whose role is active or standby, whose operational status is up or joining and who promiscuously declares, "I am the PCC coordinator". However, other election mechanisms are contemplated by the present invention. For example in another embodiment, the election criteria may take into account the capacity of each PMM or a special connectivity mechanism to a monitoring system.

Join timeout interval. This interval defines a period of time within which a PCC waits for a PMM to join, which may happen either at PCC creation, PCC merge or any PCC reconfiguration resulting from a PMM being down. An interesting situation involving this interval arises when the PMM attempting to "join" the PCC is a nominee to become the PCC coordinator. For example, assume the PMM that would "naturally" be elected PCC coordinator, i.e., the PMM with the lowest IP address, cannot consistently communicate with the other PCC members. Although it sent a message to the members inviting them to "join-up", the PMM thereafter does not contact the members within this time period. When the join timeout interval expires, the remaining members transition to a selection phase to elect another PMM as the PCC Coordinator. This prevents the cluster from "hanging" in a waiting mode for an extended period of time.

Heartbeat timer interval. This is the frequency of the transmitted PCC heartbeat messages **310** which may occur, for example, every 50 milliseconds (msecs). If a heartbeat is not detected from any PMM **400** within this interval, the remaining PMMs transition to the failure suspect threshold state and proceed as described above.

Alert timer interval. This is the frequency of a PCC alert timer which is generally greater than the regular heartbeat timer. When a PCC configuration change is pending, a PMM attempts to resolve the pending transaction by delivering cluster messages using the alert timer. PMMs transition into an alert mode when attempting to form a PCC; in this mode, they exchange messages at a faster frequency than the heartbeat in order to synchronize the members as the cluster forms.

According to another aspect of the invention, the PCC coordinator **350** further administers the common configuration by implementing a load balancing aspect of the clustering technique. Load balancing substantially optimizes the assignment of network addresses to active PMM members and is preferably implemented using the relative characteristics of each PMM and PCC service, such as PMM capacity, PCC service load and PCC service address(es). Once the PCC is "balanced", each PMM attends to client requests directed to its assigned addresses. In the event of a failure to one of their members, the active PMMs collaborate with the PCC coordinator to reassign the failed PMM's network addresses among the remaining active members.

The following Examples 1–4 illustrate the inventive load balancing technique which is preferably implemented by the PCC coordinator in accordance with a sequence of steps depicted in the flowchart of FIG. 5. Specifically, the PCC coordinator executes a novel algorithm represented by the examples and flowchart to calculate these loads for the PCC members. In the example, the PCC comprises three (3) PMMs and four (4) PCC services hosted by the PMMs. The capacity rating of the first PMM is Cap10, the capacity rating of the second PMM is Cap15 and the capacity rating of third PMM is Cap6, whereas the load rating of each service is L10, L2, L20 and L10, respectively. In addition, there are a number of service addresses (A3, A1, A5 and A2) associated with each PCC service. For example, PCC Service1 has a load rating L10 and also has 3 service addresses (A3) associated with its service; e.g., 1.1.1.1, 1.1.1.2 and 1.1.1.3.

EXAMPLE 1

Assume the following PMMs and capacities:

PMM1	Cap10
PMM2	Cap15
PMM3	Cap6

Assume the following Services with associated load ratings and addresses

Service1	L10	A3 (1.1.1.1, 1.1.1.2, 1.1.1.3)
Service2	L2	A1 (1.1.1.4)
Service3	L20	A5 (1.1.1.5, 1.1.1.6, 1.1.1.7, 1.1.1.8, 1.1.1.9)
Service4	L10	A2 (1.1.1.10, 1.1.1.11)

Referring to the flowchart of FIG. 5, the load balancing sequence starts at Step 500 and proceeds to Step 502 where the loading ratings are summed.

Sum the Service loads: 10+2+20+10=42

In Step 504, the loading rating per address for each service is calculated. For example, the Service1 has three addresses A3, so the load rating L of 10 is divided by 3 to get 3.3.

Calculate a load rating for each address for each service:
10/3=3.3, 2/1=2, 20/5=4, 10/2=5

In Step 506, an address list is created with the list being sorted, in descending order, by the load rating per address. For example, Service4 has two addresses A2 and a load rating of L10. The service load rating for each address of Service4 is 5. Thus, e.g., IP service address 1.1.1.10 is assigned a load rating per address of 5 and service address 1.1.1.11 is also assigned a load rating per address of 5. Note that an address is associated with each entry of the list.

Create an address list sorted descending by load rating per address
Service4, Load Rating 5, Service address 1.1.1.10
Service4, Load Rating 5, Service address 1.1.1.11
Service3, Load Rating 4, Service address 1.1.1.5
Service3, Load Rating 4, Service address 1.1.1.6
Service3, Load Rating 4, Service address 1.1.1.7
Service3, Load Rating 4, Service address 1.1.1.8
Service3, Load Rating 4, Service address 1.1.1.9
Service1, Load Rating 3.3, Service address 1.1.1.1
Service1, Load Rating 3.3, Service address 1.1.1.2
Service1, Load Rating 3.3, Service address 1.1.1.3
Service2, Load Rating 2, Service address 1.1.1.4

Step 506 thus calculates an anticipated load for each service address for purposes of optimally or near-optimally balancing the actual load across the entire cluster. In Step 508, the capacity ratings of the PMMs are summed.

Sum the PMM capacities: 10+15+6=31

In Step 510, the current capacity rating for each PMM is calculated and normalized to a common load unit metric (CapacityRating×TotalLoad)/TotalCapacity

Calculate the current capacity rating for each PMM normalized in Load Units: (CapacityRating*TotalLoad)/TotalCapacity
PMM1=13.5
PMM2=20.32
PMM3=8.13

In Step 512, the load ratings per address are assigned by (i) getting the first load rating by address, (ii) assigning this service address to PMM with highest current capacity rating, (iii) reducing the current capacity rating of the PMM by the load rating assigned; (iv) removing the first load rating by address from the list; and (v) repeating Step 512 until done.

Assign loading
get the first load rating by address=Service4, Load Rating 5, Service address 1.1.1.10
assign this service address to the PMM with the highest current capacity rating
PMM2=20.32
PMM2 now is responsible for service address 1.1.1.10
reduce the current capacity rating of the PMM by the load assigned
PMM2=20.32−5=15.32
The new PMM list becomes
PMM1=13.5
PMM2=15.32
PMM3=8.13
remove the first load rating by address from the list
Service4, Load Rating 5, Service address 1.1.1.11
Service3, Load Rating 4, Service address 1.1.1.5
Service3, Load Rating 4, Service address 1.1.1.6
Service3, Load Rating 4, Service address 1.1.1.7
Service3, Load Rating 4, Service address 1.1.1.8
Service3, Load Rating 4, Service address 1.1.1.9
Service1, Load Rating 3.3, Service address 1.1.1.1
Service1, Load Rating 3.3, Service address 1.1.1.2
Service1, Load Rating 3.3, Service address 1.1.1.3
Service2, Load Rating 2, Service address 1.1.1.4
repeat
get the first load rating by address=Service4, Load Rating 5, Service address 1.1.1.11
assign this service address to the PMM with the highest current capacity rating
PMM2=15.32
PMM2 now is responsible for service address 1.1.1.11
reduce the current capacity rating of the PMM by the load assigned
PMM2=15.32−5=10.32
The new PMM list becomes
PMM1=13.5
PMM2=10.32
PMM3=8.13
remove the first load rating by address from the list
Service3, Load Rating 4, Service address 1.1.1.5
Service3, Load Rating 4, Service address 1.1.1.6
Service3, Load Rating 4, Service address 1.1.1.7
Service3, Load Rating 4, Service address 1.1.1.8

Service3, Load Rating 4, Service address 1.1.1.9
Service1, Load Rating 3.3, Service address 1.1.1.1
Service1, Load Rating 3.3, Service address 1.1.1.2
Service1, Load Rating 3.3, Service address 1.1.1.3
Service2, Load Rating 2, Service address 1.1.1.4

repeat

get the first load rating by address=Service3, Load Rating 4, Service address 1.1.1.5

assign this service address to the PMM with the highest current capacity rating

PMM1=13.5

PMM1 now is responsible for service address 1.1.1.5

reduce the current capacity rating of the PMM by the load assigned

PMM1=13.5−4=9.5

The new PMM list becomes

PMM1=9.5
PMM2=10.32
PMM3=8.13

remove the first load rating by address from the list

Service3, Load Rating 4, Service address 1.1.1.6
Service3, Load Rating 4, Service address 1.1.1.7
Service3, Load Rating 4, Service address 1.1.1.8
Service3, Load Rating 4, Service address 1.1.1.9
Service1, Load Rating 3.3, Service address 1.1.1.1
Service1, Load Rating 3.3, Service address 1.1.1.2
Service1, Load Rating 3.3, Service address 1.1.1.3
Service2, Load Rating 2, Service address 1.1.1.4

repeat

get the first load rating by address=Service3, Load Rating 4, Service address 1.1.1.6

assign this service address to the PMM with the highest current capacity rating

PMM2=10.32

PMM2 is now responsible for service address 1.1.1.6

reduce the current capacity rating of the PMM by the load assigned

PMM2=10.32−4=6.32

The new PMM list becomes

PMM1=9.5
PMM2=6.32
PMM3=8.13

remove the first load rating by address from the list

Service3, Load Rating 4, Service address 1.1.1.7
Service3, Load Rating 4, Service address 1.1.1.8
Service3, Load Rating 4, Service address 1.1.1.9
Service1, Load Rating 3.3, Service address 1.1.1.1
Service1, Load Rating 3.3, Service address 1.1.1.2
Service1, Load Rating 3.3, Service address 1.1.1.3
Service2, Load Rating 2, Service address 1.1.1.4

repeat

get the first load rating by address=Service3, Load Rating 4, Service address 1.1.1.7

assign this service address to the PMM with the highest current capacity rating

PMM1=9.5

PMM1 is now responsible for service address 1.1.1.7

reduce the current capacity rating of the PMM by the load assigned

PMM1=9.5−4=5.5

The new PMM list becomes

PMM1=5.5
PMM2=6.32
PMM3=8.13

remove the first load rating by address from the list

Service3, Load Rating 4, Service address 1.1.1.8
Service3, Load Rating 4, Service address 1.1.1.9
Service1, Load Rating 3.3, Service address 1.1.1.1
Service1, Load Rating 3.3, Service address 1.1.1.2
Service1, Load Rating 3.3, Service address 1.1.1.3
Service2, Load Rating 2, Service address 1.1.1.4

repeat

get the first load rating by address=Service3, Load Rating 4, Service address 1.1.1.8

assign this service address to the PMM with the highest current capacity rating

PMM3=8.13

PMM3 is now responsible for service address 1.1.1.8

reduce the current capacity rating of the PMM by the load assigned

PMM3=8.13−4=4.13

The new PMM list becomes

PMM1=5.5
PMM2=6.32
PMM3=4.13

remove the first load rating by address from the list

Service3, Load Rating 4, Service address 1.1.1.9
Service1, Load Rating 3.3, Service address 1.1.1.1
Service1, Load Rating 3.3, Service address 1.1.1.2
Service1, Load Rating 3.3, Service address 1.1.1.3
Service2, Load Rating 2, Service address 1.1.1.4

repeat

get the first load rating by address=Service3, Load Rating 4, Service address 1.1.1.9

assign this service address to the PMM with the highest current capacity rating

PMM2=6.32

PMM2 is now responsible for service address 1.1.1.9

reduce the current capacity rating of the PMM by the load assigned

PMM2=6.32−4=2.32

The new PMM list becomes

PMM1=5.5
PMM2=2.32
PMM3=4.13

remove the first load rating by address from the list

Service1, Load Rating 3.3, Service address 1.1.1.1
Service1, Load Rating 3.3, Service address 1.1.1.2
Service1, Load Rating 3.3, Service address 1.1.1.3
Service2, Load Rating 2, Service address 1.1.1.4

repeat

get the first load rating by address=Service1, Load Rating 3.3, Service address 1.1.1.1

assign this service address to the PMM with the highest current capacity rating

PMM1=5.5

PMM1 is now responsible for service address 1.1.1.1

reduce the current capacity rating of the PMM by the load assigned

PMM1=5.5−3.3=2.2

The new PMM list becomes

PMM1=2.2
PMM2=2.32
PMM3=4.13

remove the first load rating by address from the list

Service1, Load Rating 3.3, Service address 1.1.1.2
Service1, Load Rating 3.3, Service address 1.1.1.3
Service2, Load Rating 2, Service address 1.1.1.4

repeat

get the first load rating by address=Service1, Load Rating 3.3, Service address 1.1.1.2

assign this service address to the PMM with the highest current capacity rating

PMM3=4.13

PMM3 is now responsible for service address 1.1.1.2
reduce the current capacity rating of the PMM by the load assigned
PMM3=4.13–3.3=0.83
The new PMM list becomes
PMM1=2.2
PMM2=2.32
PMM3=0.83
remove the first load rating by address from the list
Service1, Load Rating 3.3, Service address 1.1.1.3
Service2, Load Rating 2, Service address 1.1.1.4
repeat
get the first load rating by address=Service1, Load Rating 3.3, Service address 1.1.1.3
assign this service address to the PMM with the highest current capacity rating
PMM2=2.32
PMM2 is now responsible for service address 1.1.1.3
reduce the current capacity rating of the PMM by the load assigned
PMM2=2.32–3.3=–0.98 Note that negative numbers work without modification
The new PMM list becomes
PMM1=2.2
PMM2=–0.98
PMM3=0.83
remove the first load rating by address from the list
Service2, Load Rating 2, Service address 1.1.1.4
repeat
get the first load rating by address=Service2, Load Rating 2, Service address 1.1.1.4
assign this service address to the PMM with the highest current capacity rating
PMM1=2.2
PMM1 is now responsible for service address 1.1.1.4
reduce the current capacity rating of the PMM by the load assigned
PMM1=2.2–2=0.2
The new PMM list becomes
PMM1=0.2
PMM2=–0.98
PMM3=0.83
remove the first load rating by address from the list exit
The resulting solution:
PMM1 (1.1.1.5, 1.1.1.7, 1.1.1.1, 1.1.1.4)
PMM2 (1.1.1.10, 1.1.1.11, 1.1.1.6, 1.1.1.9, 1.1.1.3)
PMM3 (1.1.1.8, 1.1.1.2)

Upon completing Step 512 by exhausting the list created in Step 506, the sequence ends at Step 520 with an optimal or near optimal balance of loads across the cluster being achieved. The PCC coordinator 350 then makes the assignments by distributing the updated configuration file 340 to each member. As noted, the file 340 specifies the network service addresses that each PMM member will respond to and serve. The members store the file for future reference when servicing those addresses. For example, PMM3 receives the configuration file message indicating that it is assigned service addresses 1.1.1.8 and 1.1.1.2. Accordingly, the PMM 400 starts servicing those assigned addresses.

It should be noted that the inventive load balancing technique may be extended to allow for other considerations. For instance, the second pass through Step 512 results in addresses associated with the same service (Service4) being assigned to the same machine (PMM2). It may be desirable not to have two service addresses for the same service "clumped" to the same PMM. To avoid such clumping, the invention contemplates modification of the process to allow assignment of the second service address to the PMM with the next highest current capacity rating (e.g., PMM1 with the 13.5 current capacity rating). This modification, which is illustrated in Example 2 below, increases the availability of the PCC 300 so that in the event PMM2 fails, there will be at least one service address available for Service4 while the PCC reconfigures itself. It will be obvious to those skilled in the art that other modifications/loading balancing algorithms can be utilized to achieve the goals of improved availability, performance, fault failover, etc.

EXAMPLE 2

Assume the following PMMs and capacities:

PMM1	Cap10
PMM2	Cap15
PMM3	Cap6

Assume the following Services with associated load ratings and addresses

Service1	L10	A3 (1.1.1.1, 1.1.1.2, 1.1.1.3)
Service2	L2	A1 (1.1.1.4)
Service3	L20	A5 (1.1.1.5, 1.1.1.6, 1.1.1.7, 1.1.1.8, 1.1.1.9)
Service4	L10	A2 (1.1.1.10, 1.1.1.11)

Sum the Service loads: 10+2+20+10=42
Calculate a load rating for each address for each service: 10/3=3.3, 2/1=2, 20/5=4,10/2=5
Create an address list sorted descending by load rating per address
Service4, Load Rating 5, Service address 1.1.1.10
Service4, Load Rating 5, Service address 1.1.1.11
Service3, Load Rating 4, Service address 1.1.1.5
Service3, Load Rating 4, Service address 1.1.1.6
Service3, Load Rating 4, Service address 1.1.1.7
Service3, Load Rating 4, Service address 1.1.1.8
Service3, Load Rating 4, Service address 1.1.1.9
Service1, Load Rating 3.3, Service address 1.1.1.1
Service1, Load Rating 3.3, Service address 1.1.1.2
Service1, Load Rating 3.3, Service address 1.1.1.3
Service2, Load Rating 2, Service address 1.1.1.4
Sum the PMM capacities: 10+15+6=31
Calculate the current capacity rating for each PMM normalized in Load Units (CapacityRating*TotalLoad)/ TotalCapacity
PMM1=13.5
PMM2=20.32
PMM3=8.13
Assign loading
get the first load rating by address=Service4, Load Rating 5, Service address 1.1.1.10
assign this service address to the PMM with the highest current capacity rating
PMM2=20.32
PMM2 now is responsible for service address 1.1.1.10
reduce the current capacity rating of the PMM by the load assigned
PMM2=20.32–5=15.32
The new PMM list becomes

PMM1=13.5
PMM2=15.32
PMM3=8.13

remove the first load rating by address from the list
Service4, Load Rating **5**, Service address **1.1.1.11**
Service3, Load Rating **4**, Service address **1.1.1.5**
Service3, Load Rating **4**, Service address **1.1.1.6**
Service3, Load Rating **4**, Service address **1.1.1.7**
Service3, Load Rating **4**, Service address **1.1.1.8**
Service3, Load Rating **4**, Service address **1.1.1.9**
Service1, Load Rating **3.3**, Service address **1.1.1.1**
Service1, Load Rating **3.3**, Service address **1.1.1.2**
Service1, Load Rating **3.3**, Service address **1.1.1.3**
Service2, Load Rating **2**, Service address **1.1.1.4**

repeat

get the first load rating by address=Service4, Load Rating **5**, Service address **1.1.1.11**

assign this service address to the PMM with the highest current capacity rating
PMM2=15.32
Since PMM2 already has a Service4 Service address, select the next highest
PMM1=13.5
PMM1 now is responsible for service address **1.1.1.11**

reduce the current capacity rating of the PMM by the load assigned
PMM1=13.5−5=8.5
The new PMM list becomes
PMM1=8.5
PMM2=15.32
PMM3=8.13

remove the first load rating by address from the list
Service3, Load Rating **4**, Service address **1.1.1.5**
Service3, Load Rating **4**, Service address **1.1.1.6**
Service3, Load Rating **4**, Service address **1.1.1.7**
Service3, Load Rating **4**, Service address **1.1.1.8**
Service3, Load Rating **4**, Service address **1.1.1.9**
Service1, Load Rating **3.3**, Service address **1.1.1.1**
Service1, Load Rating **3.3**, Service address **1.1.1.2**
Service1, Load Rating **3.3**, Service address **1.1.1.3**
Service2, Load Rating **2**, Service address **1.1.1.4**

repeat

get the first load rating by address=Service3, Load Rating **4**, Service address **1.1.1.5**

assign this service address to the PMM with the highest current capacity rating
PMM2=15.32
PMM2 now is responsible for service address **1.1.1.5**

reduce the current capacity rating of the PMM by the load assigned
PMM2=15.32−4=11.32
The new PMM list becomes
PMM1=8.5
PMM2=11.32
PMM3=8.13

remove the first load rating by address from the list
Service3, Load Rating **4**, Service address 1.1.1.6
Service3, Load Rating **4**, Service address 1.1.1.7
Service3, Load Rating **4**, Service address **1.1.1.8**
Service3, Load Rating **4**, Service address **1.1.1.9**
Service1, Load Rating **3.3**, Service address **1.1.1.1**
Service1, Load Rating **3.3**, Service address **1.1.1.2**
Service1, Load Rating **3.3**, Service address **1.1.1.3**
Service2, Load Rating **2**, Service address 1.1.1.4

repeat

get the first load rating by address=Service3, Load Rating **4**, Service address 1.1.1.6

assign this service address to the PMM with the highest current capacity rating
PMM2=11.32
Since PMM2 already has a Service3 Service address, select the next highest
PMM1=8.5
PMM1 now is responsible for service address **1.1.1.6**

reduce the current capacity rating of the PMM by the load assigned
PMM1=8.5−4=4.5
etc.

A further extension to the basic load balancing technique is provided in Example 3 below. Referring to the flowchart of FIG. **5**, an additional step (such as Step **514** prior to Step **520**) may include the following action: If load balancing with existing service address connections, then reduce the current capacity rating for each PMM in Step **510** by the loads of the service addresses already assigned, make sure that the loads just assigned are not in the address list, then proceed to step **502**. The purpose of Step **514** is to rebalance without moving any existing assignments, even if the reconfiguration results in a less-optimal balance. In other words, the intent is to allow the PMMs to continue to properly service the existing connections and only "reload balance" those service addresses that require redistribution because of the loss/failure of a PMM. This alternate embodiment of the inventive balancing method pertains to an established PCC cluster **300** with a suspect PMM that thereby necessitates redistribution and rebalacing of load addresses.

EXAMPLE 3

Assume the final configuration of PMMs and service addresses in Example 1.
PMM1 (1.1.1.5, 1.1.1.7, **1.1.1.1**, 1.1.1.4)
PMM2 (1.1.1.10, 1.1.1.11, 1.1.1.6, 1.1.1.9, 1.1.1.3)
PMM3 (1.1.1.8, 1.1.1.2)

The Service Address configuration is the same:
Service4, Load Rating **5**, Service address **1.1.1.10**
Service4, Load Rating **5**, Service address **1.1.1.11**
Service3, Load Rating **4**, Service address **1.1.1.5**
Service3, Load Rating **4**, Service address **1.1.1.6**
Service3, Load Rating **4**, Service address **1.1.1.7**
Service3, Load Rating **4**, Service address **1.1.1.8**
Service3, Load Rating **4**, Service address **1.1.1.9**
Service1, Load Rating **3.3**, Service address **1.1.1.1**
Service1, Load Rating **3.3**, Service address **1.1.1.2**
Service1, Load Rating **3.3**, Service address **1.1.1.3**
Service2, Load Rating **2**, Service address **1.1.1.4**

The original PMM capacity configuration is the same:
PMM1=13.5
PMM2=20.32
PMM3=8.13

Assume that PMM1 has gone down and a reload is necessary
PMM1 (1.1.1.5, 1.1.1.7, 1.1.1.1, **1.1.1.4)**
PMM2 (1.1.1.10, 1.1.1.11, 1.1.1.6, 1.1.1.9, 1.1.1.3)
PMM3 (1.1.1.8, 1.1.1.2)

Load Balance

normalize the new PMM list
Service4, Load Rating **5**, Service address **1.1.1.10**
Service4, Load Rating **5**, Service address **1.1.1.11**
Service3, Load Rating **4**, Service address **1.1.1.5**
Service3, Load Rating **4**, Service address **1.1.1.6**
Service3, Load Rating **4**, Service address **1.1.1.7**
Service3, Load Rating **4**, Service address **1.1.1.8**
Service3, Load Rating **4**, Service address **1.1.1.9**

Service1, Load Rating 3.3, Service address 1.1.1.1
Service1, Load Rating 3.3, Service address 1.1.1.2
Service1, Load Rating 3.3, Service address 1.1.1.3
Service2, Load Rating 2, Service address 1.1.1.4

PMM2=20.32-5-5-4-4-3.3=-0.98

Service4, Load Rating 5, Service address 1.1.1.10
Service4, Load Rating 5, Service address 1.1.1.11
Service3, Load Rating 4, Service address 1.1.1.5
Service3, Load Rating 4, Service address 1.1.1.6
Service3, Load Rating 4, Service address 1.1.1.7
Service3, Load Rating 4, Service address 1.1.1.8
Service3, Load Rating 4, Service address 1.1.1.9
Service1, Load Rating 3.3, Service address 1.1.1.1
Service1, Load Rating 3.3, Service address 1.1.1.2
Service1, Load Rating 3.3, Service address 1.1.1.3
Service2, Load Rating 2, Service address 1.1.1.4

PMM3=8.13-4-3.3=0.83

The new PMM list is:
PMM2=-0.98
PMM3=0.83

Note that if the cluster of PMMs keep track of their loads dyamically, these computations may be done with the statistics gathered during real-time rather than the static values. As well, if one or more PMMs discover that the dynamic loads being gathered in real-time exceed the static loads by some amount (e.g., a percentage) then the affected PMM 400 may request a rebalance from the PCC Coordinator 350.

The new Service Address list is (assignments to PMM1):
Service3, Load Rating 4, Service address 1.1.1.5
Service3, Load Rating 4, Service address 1.1.1.7
Service1, Load Rating 3.3, Service address 1.1.1.1
Service2, Load Rating 2, Service address 1.1.1.4

balance the new list

get the first load rating by address=Service3, Load Rating 4, Service address 1.1.1.5

assign this service address to the PMM witb tbe highest current capacity rating
PMM3=0.83
PMM3 now is responsible for service address 1.1.1.5

reduce the current capacity rating of the PMM by the load assigned
PMM3=0.83-4=-3.17
The new PMM list becomes
PMM2=-0.98
PMM3=-3.17

remove tbe first load rating by address from the list
Service3, Load Rating 4, Service address 1.1.1.7
Service1, Load Rating 3.3, Service address 1.1.1.1
Service2, Load Rating 2, Service address 1.1.1.4

repeat

get the first load rating by address Service3, Load Rating 4, Service address 1.1.1.7

assign this service address to the PMM with the highest current capacity rating
PMM2=-0.98
PMM2 now is responsible for service address 1.1.1.7

reduce the current capacity rating of the PMM by the load assigned
PMM2=-0.98-4=-4.98

The new PMM list becomes
PMM2=-4.98
PMM3=-3.17

remove the first load rating by address from the list
Service1, Load Rating 3.3, Service address 1.1.1.1
Service1, Load Rating 3.3, Service address 1.1.1.2

repeat

get the first load rating by address=Service1, Load Rating 3.3, Service address 1.1.1.1

assign this service address to the PMM with the bighest current capacity rating
PMM3=-3.17
PMM3 now is responsible for service address 1.1.1.1

reduce tbe current capacity rating of the PMM by the load assigned
PMM3=3.17-3.3=-6.47
The new PMM list becomes
PMM2=-4.98
PMM3=-6.47

remove tbe first load rating by address from the list
Service2, Load Rating 2, Service address 1.1.1.4

repeat

get the first load rating by address=Service2, Load Rating 2, Service address 1.1.1.4

assign this service address to the PMM witb the highest current capacity rating
PMM2=-4.98
PMM2 now is responsible for service address 1.1.1.4

reduce the current capacity rating of tbe PMM by the load assigned
PMM2=-4.98-2=-6.98
The new PMM list becomes
PMM2=-6.98
PMM3=-6.47

exit

The resulting solution:
PMM2 (1.1.1.10, 1.1.1.11, 1.1.1.6, 1.1.1.9, 1.1.1.3, 1.1.1.7, 1.1.1.4)
PMM3 (1.1.1.8, 1.1.1.2, 1.1.1.5, 1.1.1.1)

If a more optimal balance can be achieved because the results of the newly balanced configuration are "suspicious", e.g., the spread between PMM2 loading and PMM3 loading is undesirable or service addresses are "clumped", tben the inventive technique allows choosing among one or more of the service addresses assigned to PMM2 and/or PMM3, adding them to the list of service addresses to be rebalanced and repeating Example 3.

As a last example, assume that a new PMM or more PCC service addresses are added to a PCC 300 resulting in the need to rebalance. A rebalance may be tempered by the desire to move as few previously assigned service addresses as possible to prevent a temporary loss of service to clients using the services.

EXAMPLE 4

do loading as per Example 3 (with or without clumping prevention);

don't commit the configuration to the PCC yet;

for all combinations:

take a look at spread and see if there is some movement of a Service Address that would improve the balance;

check for address distribution violation (clumping);

when all done commit to PCC.

The goals of the examples illustrated above are to produce a reasonable balance of loads across the PCC while maintaining continuity of service, even at the expense of slight sub-optimal balance.

As noted, the proxy cache clustering technique described herein provides a fault-failover mechanism for the PCC. In the event of a failure to a PMM **400**, the PCC coordinator **350** proceeds to rebalaoce the load over the remaining members of the PCC (Examples 1–4) and then continues normal operation under the rebalanced configuration. Clients without open connections to the failed PMM function as though no failure event had occurred. For instance, a browser **110** requesting service at an address of a PMM that is still "up" does not experience an interrupt in service. Clients with open connections to the failed PMM may, however, time out, retry and continue communication without further interruption. At this point, the PCC **300** and clients **120** have fully recovered from the fault.

In summary, the inventive proxy cache cluster (PCC) cooperates with a service provider of a communications network to increase the availability of services offered by the provider to clients connected to the network. By functioning as a system of proxy cache servers defined by a common configuration of PMMs, hosted PCC services and static PCC configuration parameters, the PCC improves availability, performance and scalability of the service provider by, e.g., moving the network addresses serviced by a failed PMM to surviving PMMs of the PCC according to the inventive proxy cache clustering technique.

Scalability of the service provider may be further achieved by adding PMMs, PCC service addresses, and/or PMM network addresses to the PCC. Adding PCC service (IP) addresses means, for instance, adding a new HTTP service address, whereas adding PMM network (IP) addresses means adding a new PMM which has a network address. It should be noted that a situation where scaling does not benefit by the addition of a PMM is when there are fewer PCC service addresses than PMMs. A service address can not be split across a PMM, thus the minimum configuration is a one-to-one relationship between PMMs and PCC service addresses. Although there may be situations where additional PMMs may outnumber the service addresses in a PCC, the only use for these "spare" PMMs is for failover in the event of a faulty PMM.

The minimum configuration of the PCC preferably comprises at least two PMMs, although there is no architectural limit to the number of PMMs that can be assigned to a PCC. The minimum configuration does not lend itself to load balancing in the event of one PMM failing, but if a failure occurs, there is automatic fail-over to the surviving PMM. More than two PMMs is desirable because load balancing can be performed in the event of a failure. In addition, the illustrative configuration of the service provider/web site includes a plurality of web servers **200**, each having a PMM resident thereon. Collectively, the PMMs function as front-end entities to "float" all of the network addresses of the service provider **180**. Thus if one of the web servers **200** fails or otherwise goes down, its network address is reassigned other PMMs, as described herein.

While there has been shown and described an illustrative embodiment for increasing the availability of services offered by a service provider to clients connected to a communications internetwork in accordance with an inventive proxy cache clustering technique, it is to be understood that various other adaptations and modifications may be made within the spirit and scope of the invention. For example, according to another aspect of the invention, the novel clustering technique provides fault-failover partitioning of a PCC. A desirable artifact of the invention is that a single PCC may split into more than one PCC. If certain members of the PCC lose contact with the rest of the PCC members, the portion of the PCC that does not have a PCC coordinator proceeds to elect a new PCC coordinator and form a new PCC. The PCC with the original PCC coordinator then load balances across the remaining PMMs and continues servicing assigned addresses. This activity occurs despite the number of fragments that a PCC may be partitioned into.

In yet another aspect of the present invention, fault-correction partition merging is provided by the invention. Continuing with the example above, assume the lost portion of the PCC resumes contact with the rest of the PCC members and attempts to rejoin the cluster. An immediate concern is which PCC coordinator becomes the coordinator of the reformed PCC. The invention contemplates determining the busiest cluster and, thereafter, not disturb it so as to minimize perturbation to service. According to an embodiment of the invention, the busiest cluster is determined by examining a most recent service interval (contained in the heartbeat message **310**) and the PCC coordinator with the lowest interval becomes the new coordinator.

A service interval is the interval between the last service request and the current service request, e.g., from a browser **110** to a web site **180**. The PCC coordinator with the lowest service interval (e.g., 30 ms vs. 500 ms) is assumed to be servicing more traffic so, according to this embodiment, it is elected the new coordinator. Note that other metrics can also be used to determine the new coordinator. The losing coordinator then broadcasts a shutdown restart message to all members of the losing cluster. The new coordinator proceeds to integrate the new PMMs by load balancing according to the clustering technique described herein. The shutdown restart message instructs the PMMs to finish their current work and then transition into a joining mode. The surviving PCC coordinator then balances across those PMMs that are joined.

The foregoing description has been directed to specific embodiments of this invention. It will be apparent, however, that other variations and modifications may be made to the described embodiments, with the attainment of some or all of their advantages. Therefore, it is the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the invention.

What is claimed is:

1. A method for increasing the availability of services offered by a service provider to clients connected to a communications network, the clients accessing the services by issuing requests to service addresses associated with these services, the method comprising the steps of:

providing a plurality of processor/memory mechanisms (PMMs) adapted to cooperatively interact in order to receive and service the requests on behalf of the service provider;

organizing the PMMs as one or more proxy cache clusters (PPCs) by dynamically assigning each PMM to one or more PCCs;

balancing the service addresses among PMMs by assigning selected service addresses to each PMM of the PCC; and

rebalancing the service addresses among PMMs of the PCC in response to dynamic changes in the PCC.

2. The method of claim **1** wherein the step of organizing comprises the step of configuring each PMM with a unique identifier (ID), a network address and PCC configuration software.

3. The method of claim **2** wherein the step of organizing further comprises the step of notifying each PMM of avail-

ability in the PCC using a heartbeat message broadcasted to each member of the PCC.

4. The method of claim 3 wherein the step of organizing further comprises the step of sharing a common configuration among the PMMs that describes the PCC in terms of the PMMs, static PCC configuration parameters and hosted PCC services.

5. The method of claim 4 wherein the step of sharing a common configuration comprises the step of characterizing each hosted PCC service by a load rating value that reflects a measure of the PCC service's resource consumption.

6. The method of claim 4 wherein the step of sharing a common configuration comprises the step of characterizing the hosted PCC services by service type.

7. The method of claim 6 wberein the service type comprises one of a Hypertext Transfer Protocol (HTTP) proxy service, an HTTP accelerator service, a file transfer protocol proxy service, real audio and real video.

8. The method of claim 4 wherein the step of sharing a common configuration comprises the step of characterizing the hosted PCC services by service type parameters that are unique to each service.

9. The method of claim 5 wherein the step of providing comprises the step of characterizing each PMM by a capacity rating value that reflects a measure of the PMM's ability to provide resources.

10. The method of claim 1 wherein the step of balancing the service addresses comprises the step of designating a particular PMM to function as a PCC coordinator to assign the service addresses to the PMMs.

11. The method of claim 10 wherein the step of balancing the service addresses further comprises the step of periodically distributing a message containing the service address assignments from the PCC coordinator to the PMMs.

12. The method of claim 11 wherein the step of periodically distributing comprises the step of uniquely identifying the service address assignment message as issued by the PCC coordinator by marking the message witb a stamp.

13. The method of claim 11 wherein the step of providing comprises the step of characterizing each PMM by a role that describes various activities of the PMM as manifested in one of the service address assignment message and the heartbeat message.

14. The method of claim 13 wherein the role comprises an active role in one or more PCCs tbat allows the PMM to participate in all PCC activities.

15. The method of claim 13 wherein the role comprises an standby role in one or more PCCs that prohibits assignments to the PMM until at least one other PMM in tbe PCC assumes one of a down or unusable role.

16. The method of claim 15 wherein the unusable role indicates that the PMM cannot be used by the PCC, whereas the down role is automatically assigned by the PCC coordinator upon failure to detect a heartbeat message within a predetermined period.

17. The method of claim 13 wherein the role comprises an active role in one or more PCCs that allows the PMM to participate in all PCC activities and a standby role in one or more PCCs that prohibits assignments to the PMM until at least one other PMM in the PCC assumes one of a down or unusable role.

18. The method of claim 10 wherein the step of providing comprises the step of characterizing each PMM by a primary communication network address for communicating among the PMMs and the PCC coordinator.

19. The method of claim 18 wherein the primary communication network address is the unique ID of the PMM.

20. The method of claim 1 wherein the step of providing comprises the step of characterizing each PMM by an operational status including one of a joining status when the PMM attempts to rejoin the PCC after being offline, an up status after the PCC coordinator accepts the PMM and rebalances the PCC, a leaving status when the PMM becomes unavailable to tbe PCC and a down status after the PMM is unavailable to tbe PCC.

21. The method of claim 1 wherein the step of balancing comprises the steps of:

summing the load ratings of the hosted PCC services;

calculating tbe load rating per address for eacb hosted PCC service;

creating an address list that is sorted, in decending order, by the calculated load rating per address;

summing the capacity ratings of the PMMs; and

calculating a current capacity rating of each PMM normalized to a common load unit metric.

22. The method of claim 1 wherein dynamic changes in the PCC comprises addition or deletion of services.

23. The method of claim 1 wherein dynamic changes in the PCC comprises real-time collection of load data.

24. The method of claim 1 wherein dynamic changes in the PCC comprises failure of a PMM.

25. The method of claim 1 wherein the step of rebalancing comprises tbe steps of:

summing the load ratings of the hosted PCC services;

calculating the load rating per address for each hosted PCC service;

creating an address list that is sorted, in decending order, by the calculated load rating per address;

summing the capacity ratings of the PMMs; and

calculating a current capacity rating of each PMM normalized to a common load unit metric.

26. The method of claim 1 wherein the step of organizing comprises the step of dynamically establishing one or more new clusters that continue to service clients if one or more PMMs loses contact with their original clusters.

27. A computer-readable medium comprising: instructions and data written thereon, said instructions and data containing information for the practice of the method of the claim 1.

28. Electromagnetic signals traveling over a computer network comprising: said electromagnetic signals carrying information for the practice of the method of claim 1.

29. A system for increasing the availability of services offered by a service provider to clients connected to a communications network, the clients accessing the services by issuing requests to network addresses associated witb these services, the system comprising:

a plurality of processor/memory mechanisms (PMMs) cooperatively interacting to receive and service the requests on behalf of the service provider, each PMM characterized by a primary communication address and assigned at least one network address to service;

means for organizing the PMMs as one or more proxy cache clusters (PPCs) by dynamically assigning each PMM to one or more PCCs; and

means for balancing service addresses among PMMs by assigning selected service addresses to each PMM of the PCC

means for rebalancing service addresses among PMMs of the PCC in response to dynamic changes in the PCC.

* * * * *